Shallow Neural Networks for Autonomous Robots

Mariela De Lucas Alvarez

Ocean Systems Laboratory

School of Engineering and Physical Sciences

Heriot-Watt University

A thesis submitted for the degree of

Doctor of Philosophy

June 2021

Abstract

The use of Neural Networks (NNs) in modern applications is already well established thanks to the technological advancements in processing units and Deep Learning (DL), as well as the availability of deployment frameworks and services. However, the embedding of these methods in robotic systems is problematic when it comes to field operations. The use of Graphics Processing Units (GPUs) for such networks requires high amounts of power which would lead to shortened operational times. This is not desired since autonomous robots already need to manage their power supply to accommodate the lengths of their missions which can extend from hours to days. While external processing is possible, real-time monitoring can become unfeasible where delays are present. This also applies to autonomous robots that are deployed for underwater or space missions.

For these reasons, there is a requirement for shallow but robust NN-based solutions that enhance the autonomy of a robot. This dissertation focuses on the design and meticulous parametrization complemented by methods that explain hyper-parameter importance. This is performed in the context of different settings and problems for autonomous robots in field operations.

The contribution of this thesis comes in the form of autonomy augmentation for robots through shallow NNs that can potentially be embedded in future systems carrying NN processing units. This is done by implementing neural architectures that use sensor data to extract representations for event identification and learn patterns for event anticipation. This work harnesses Long Short-Term Memory networks (LSTMs) as the underpinning framework for time series representation and interpretation. This has been tested in three significant problems found in field operations: hardware malfunction classification, survey trajectory classification and hazardous event forecast and detection.

In loving memory of my grandmother, Etelvina.

Acknowledgments

I would like to first thank my supervisor Prof. David Lane for the opportunity to be part of the Ocean Systems Lab at Heriot Watt University and his support into taking my own research paths. I also would like to thank my advisors Prof. Keith Brown and Prof. Helen Hastie. Special thanks go to Dr. Kelvin Hamilton from Seebyte for facilitating information and platform on RECOVERY and to Dr. Xingkun Liu for his assistance using the REGIME data. A word of gratitude to Mr. Len McLean for always providing help around the laboratory.

I am grateful to Dr. Frank Kirchner for giving me the opportunity to be a guest researcher at the Robotics Innovation Center at the German Research Center for Artificial Intelligence and to Dr. Elsa Kirchner for welcoming me into her team during this period. I also would like to thank Elmar Berghöfer for his encouragement and guidance. Special thanks to Raúl Dominguez for his assistance in collecting the data for the AsguardIV rover.

Also to the friends and acquaintances I made at the Ocean Systems Lab, Robocademy and RIC, thank you for your company and laughs we shared. It was these small moments that made this academic period so memorable. Finally, to my parents and sister for their constant encouragement and to my best friend Vic for his unconditional emotional support.

Research Thesis Submission. Please note this form should be bound into the submitted thesis.

Name:	*Mariela De Lucas Alvarez*		
School:	Engineering and Physical Sciences		
Version: *(i.e. First, Resubmission, Final)*	Final	Degree Sought:	*PhD*

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1. The thesis embodies the results of my own work and has been composed by myself
2. Where appropriate, I have made acknowledgement of the work of others
3. The thesis is the correct version for submission and is the same version as any electronic versions submitted*.
4. My thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5. I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.
6. I confirm that the thesis has been verified against plagiarism via an approved plagiarism detection application e.g. Turnitin.

ONLY for submissions including published works

Please note you are only required to complete the Inclusion of Published Works Form (page 2) if your thesis contains published works)

7. Where the thesis contains published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) these are accompanied by a critical review which accurately describes my contribution to the research and, for multi-author outputs, a signed declaration indicating the contribution of each author (complete)
8. Inclusion of published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) shall not constitute plagiarism.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate:	[illegible]	Date:	10.06.2021

Submission

Submitted By *(name in capitals)*:	Mariela De Lucas Alvarez
Signature of Individual Submitting:	[illegible]
Date Submitted:	10.06.2021

For Completion in the Student Service Centre (SSC)

Limited Access	Requested	Yes		No		Approved	Yes		No	
E-thesis Submitted (***mandatory for final theses)***									
Received in the SSC by *(name in capitals)*:					Date:				

Contents

List of Figures

List of Tables

Glossary

AdaGrad	Adaptive Gradient Algorithm
Adam	Adaptive Moment Estimation
AGV	Autonomous Ground Vehicle
AUV	Autonomous Underwater Vehicle
AUC-ROC	Area Under the Receiver Operating Characteristics
AI	Artificial Intelligence
AIC	Akaike Information Criterion
ASIC	Application Specific Integrated Circuit
ANN	Artificial Neural Network
BO	Bayesian Optimization
BIC	Bayesian Information Criterion
CNN	Convolutional Neural Network
CV	Cross-Validation
DL	Deep Learning
DNN	Deep Neural Network
DFKI	German Research Center for Artificial Intelligence
EI	Expected Improvement
EM	Expectation Maximization
FCN	Fully Connected Network
FDAS	Fault Diagnosis and Accomodation System
GMM	Gaussian Mixture Model
GP	Gaussian Process
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GPS	Global Positioning System
HB	Hyperband
HRI	Human Robot Interaction
HBBO	Hyperband Bayesian Optimization
HMM	Hidden Markov Model
IC	Information Criteria
IMU	Inertial Measurement Unit
IMM	Interactive Multiple Model

KDE	Kernel Density Estimator
LSTM	Long Short-Term Memory Network
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NN	Neural Network
PID	Proportional–Integral–Derivative
PR	Precision-Recall
PCA	Principal Component Analysis
RAS	Robotics and Autonomous Systems
RBFN	Radial Basis Function Network
ReLU	Rectified Linear Unit
RIC	Robotics Innovation Center
RNN	Recurrent Neural Network
RMSProp	Root Mean Square Propagation
ROC	Receiver Operating Characteristic
SH	Successive Halving
SHAP	Shapley Additive Explanations
SM	System Monitoring
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SOM	Self-Organizing Map
TPE	Tree Parzen Estimator
TPU	Tensor Processing Unit
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

"Never send a human to do a machine's job."

-Agent Smith, *The Matrix*

Persistent autonomy is a requirement for robot mission execution in unknown or dangerous environments. This concept refers to the goal of extending the capacity of an autonomous robot to be uncontrolled for extended periods of time, i.e. without the need to have a human interfering to solve an emergent complication that the robot cannot solve on its own (Lane et al. 2012). Its implementation is hardly trivial and demands that an essential consideration is taken into account from the very start - understanding or interpreting the data before prescribing a methodology.

Firstly, the implications of the requirements to automate a certain action needs to be understood. There are many ways to observe this based on the level of action that is needed, e.g. operational, tactical or strategic. This at the same time determines the engineering degree of the input to a specific algorithm when taking information from the environment in order to maintain a constant updated knowledge of world and system so that the robot can act upon it.

In this thesis, data analysis techniques in combination with neural networks are employed to formulate solutions from problems that arise in the field of survey robotics that require persistent autonomy. In the following sections the motivation for this work, the challenges, scope and contributions are presented.

1.1 Motivation: Neural Network-Enabled Autonomy

This dissertation was done within the European training and research framework Robocademy: European Academy for Marine and Underwater Robotics. The action line that is followed here, Autonomy, called for the development of methods that extend or support autonomous capabilities of unmanned vehicles.

How can the level of autonomy in exploration robots be extended? This scientific debate has been expanded in a wide span of research fields. It has been fostered from the vast tasks and applications in which robots are deployed and the need for easier, safer and, in many cases, cost-effective mission execution. Presently, many tasks that are carried out by exploration vehicles are significantly monitored and intervened by a human operator or expert, i.e. conditionally automated. The concept of autonomy has been widely discussed in the Artificial Intelligence community (Russell et al. 2014). As it pertains to robots, a strong definition of autonomy is presented in Beer et al. 2014.

Autonomy The extent to which a robot can sense its environment, plan in accordance, and act upon it with the purpose of completing some goal (either given to or created by the robot) without external intervention.

The stride to increase autonomous capabilities has progressed with various perspectives in monitoring of the environment and the vehicle itself. A taxonomy for degrees of autonomous driving clearly defines the capability boundaries a vehicle must surpass to achieve a rank (SAE International 2014). These have been originally established as driving automation standards but can be easily applicable to unmanned vehicles from other domains. The key features that contrast conditional from high to full automation lie in the state monitoring and human involvement in contingency performance. These capabilities on their own provide valuable enhancements but in conjunction with embedded planning these qualities can be used for context-triggered adaptive behaviour to furthermore increase the degree of autonomy.

One may talk about different subsystems that enable autonomous capabilities of a robot to have controlled mobility, localization, perception of its environment, etc. Individually, these capabilities can have their own assigned fields of study and their own unresolved research questions. It is safe to say that all these capabilities provide a degree of autonomy to an agent, however, the discussion of long-term autonomy in this thesis is rather geared towards the enabling information processing components of an autonomous agent that provide it with enough self introspection so that it may perform whatever task it has been commended to do with the least possible aid from a human operator.

For autonomy to be exhibited, the robot or vehicle must have a prior knowledge of its state in order for it to act upon it. Consequently, it can be fit to update or correct its current state (Fig 1.1). How is this corrective action triggered? For this, the vehicle needs to have context awareness.

Context is the set of environmental circumstances, both favorable and unfavorable, that surround an event. In the examination of software/hardware performance, context comprises the operational environment and its interactions with other systems or elements.

While this is a definition that can generalize to any problem, in this thesis the contexts will consist of tasks performed by autonomous survey robots and operational events present

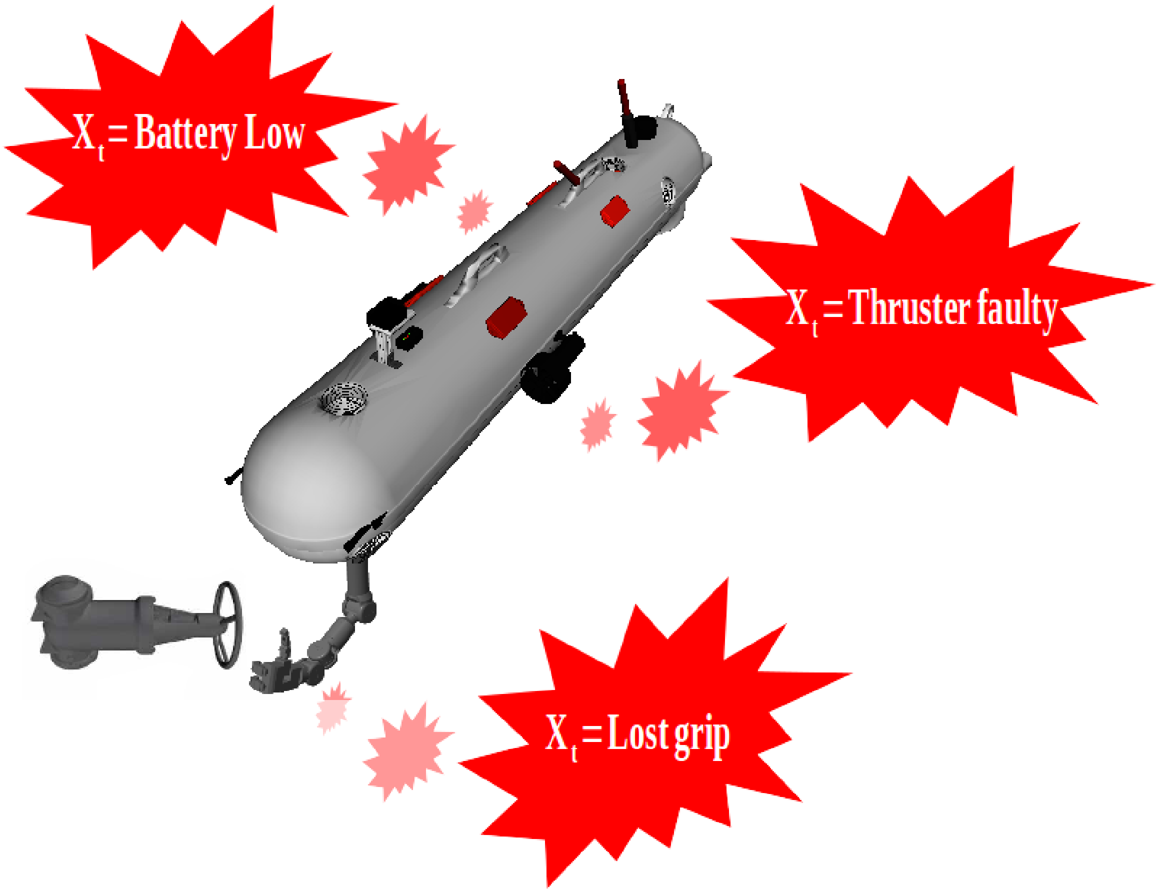


Figure 1.1: Some context awareness examples in an Autonomous Underwater Vehicle (AUV).

during the execution of their tasks. To define context, data needs to be analyzed as a series of consecutive instances in time and not as an isolated data snapshot, i.e. data streams contain temporal dependencies that collectively represent specific behaviours or characteristics.

With the need for minimizing human intervention, autonomous robots need context awareness to overcome the various unforeseen situations that arise every time they are deployed into an uncontrolled environment. Three essential motives for identifying contexts are identified:

1. **System failures.** Such can range from loss of communication of the vehicle to hardware malfunction or deterioration.

2. **Uncertainty.** An inherent quality of dangerous, unknown or inaccessible environments.

3. **External exceptions.** Are abnormal occurrences in the cell environment that may cause execution failures.

This thesis examines neural network models for time series analysis concentrating in state monitoring for detecting and forecasting. Through this, the vehicle is enabled with introspection capacities that facilitate the implementation of adaptive behaviours such as understanding the expected outcomes of a task, recognizing anomalies along the way and diagnosing its status.

1.2 Challenges

From a deployment perspective, it is currently extremely restrictive to implement large Deep Learning (DL) models due to a number of reasons. The processing power of the specialized processing units (Graphics Processing Units (GPUs)) that train and evaluate these models have high energy requirements. Such are not compatible with the power efficiency targets on which most autonomous survey robots are designed and built. With this in mind, two directions are available. Modern battery technology needs to progress accordingly in order to embed DL with the robots or the implementation of shallow Neural Networks (NNs), which integrate 1, 2, or 3 levels of non-linear operations (Bengio 2009), needs to be explored to suit these needs. While the first option will result in many benefits as a whole to field operations, the second is the challenge addressed in this work.

Another point of view to circumvent this matter is to transmit data on-line to a centralized processing unit not embedded in the robot. While this is a possibility for some applications, connectivity is subject to domain limitations where range and medium dictate the communication mechanisms. This is a fundamental characteristic of underwater and space exploration missions. If implementing a model that requires high processing power, a feasible solution is to have the data processed off-line or transmitted on-line for processing device to a control station.

This leads to other considerations from a technical perspective. Embedding the neural network models on-board the vehicles could potentially be feasible if the models were trained off-line and the architecture was small enough to allow for on-line processing without significantly diminishing the robustness of diagnosing the problem at hand. This forces us to be especially judicious with the manufacturing of the data.

When analyzing time-series within a supervised learning scheme, hand-crafting samples is expensive in terms of time and effort especially if there are large amounts of data. Data manufacturing, in this case, refers to the process of labeling and segmenting data to best represent or describe the context of interest.

Hence, as in any machine learning problem, data inspection needs to come first. Traditional statistical methods can provide us an initial perspective and bring insight into the definition of the problem, e.g. are there trends manifesting? is there missing data? is the data noisy? - and so on. Data selection is also very important, i.e. for not including unnecessary or redundant data, as this could lead to potentially learning the wrong attribute, take an unnecessary amount of time to train, etc. Then it must be defined if the problem is to be a supervised or an unsupervised learning problem.

In supervised learning, there are known categories from the data and that way a ground truth is available through direct observation. An unsupervised or semi-supervised learning problem could arise from certain cases. One is the need to identify certain contexts that arise in a sporadic or unexpected manner making examples hard to come by. A different situation can present itself when the data manifests a similitude of statistical characteristics within a set of distinct contexts.

The following section defines the examined scenarios and context definitions and their interpretations within the field of persistent autonomy for survey robotics.

1.3 Examined Scenarios

Autonomous robots face different challenges according to their domain of deployment. The underwater domain requires communication methods that are reliable in traversing water instead of air. Trajectory execution and motion models need to account for uncertain mobility patterns and unforeseen faults. On the ground domain communications can be stifled in subterranean depths. Deployment and retrieval also pose a challenge as the medium can be hazardous for the human operators in either domain. Two relevant examples can be given for separate domains. In the marine area, the case of an AUV deployed in Monterey Bay, California, where a white shark attacked the Tethys AUV within thirteen minutes of deployment (Stanway et al. 2015).

Similar to underwater applications, the space and land domain also challenging to human exploration and requires autonomous agents to undertake many tasks. Being a remote domain, it shares all of the threats the underwater domain carries to an increased level as deployment and retrieval of the autonomous vehicles cannot generally be performed by humans. Therefore, they need to comprise autonomous capabilities to execute missions successfully. Consider the Fukushima-Daiichi nuclear plant accident in 2011. Operators based outside the building attempted to suppress the critical conditions occurring inside the building. As a consequence, it was recognized that disaster unmanned ground robots needed to be designed to replace human operators in hazardous circumstances (Kawatsuma et al. 2012).

Regardless of the area of application, system and mission monitoring can also be challenging in unknown and inaccessible environments. Faults and anomalies are hard to simulate and are so varied in nature that when they do occur it is complex to generate models that exhaustively classify them.

This work considers the emerging behaviours that arise when the autonomous vehicles are deployed in survey missions in the marine and space domain. A range of behaviours arise in these scenarios. These behaviours can take the form of:

- A set of execution characteristics.
- A set of tasks or events during execution.

- A set of properties characterizing an event that define a state or condition.

When deployed, autonomous systems and robots handle a wide range of information either in quantity or variety for interpretation and actuation. The context therefore is subject to the application and domain in which their functions occur. In this work, three categories of context are explored:

1. Hardware-oriented context
2. Task-oriented context
3. Event-oriented context

Hardware-oriented context focuses on identifying functioning and malfunctioning context-specific states of electronic components or systems. Much of the interest is focused in observing the data to monitor adequate operation. While standard rule-based approaches can provide basic reliable information, it can become a complex task to establish rules as new states are identified with similar ranges of data values. This work focuses in the monitoring and diagnosis of thruster malfunction in an AUV.

Task-oriented context focuses on identifying the state of a currently executed task. While this can be generic and a task can take many forms, this work defines task as an instruction an autonomous vehicle must execute, specifically navigational trajectories for survey and inspection. The importance of understanding these actions as such is not only to furnish the human operator with understandable information but to also set knowledge foundations on which the robot can reason for instance on mission health and triggering recovery actions. This work focuses on navigational trajectory classification for an AUV.

Event-oriented context differs from task-oriented in the nature of the categorization. Events are defined as classes that fall out of the predefined procedures the robot has to execute. They represent behaviours that in constitution are not malfunctions but diverge from expected or normal behaviour that are circumstantial. Identifying events is valuable for alerting systems of imminent or undesired outcomes. Consequently, event-oriented context detection is geared towards safeguarding the integrity of the autonomous vehicle and supporting the cohesion of the mission. This work focuses in mobility hazard forecast and detection for a space rover.

1.4 Contributions

These three contexts all share high scientific relevance and their exploration and development in this dissertation present the following contributions.

- **High-performing shallow architectures.** The design of shallow neural networks that use relevant input to reduce training and processing time without compromising robust performance for classification or prediction of states. This is achieved for three different problems specific to autonomous robots in field operations.

1. **Hardware-oriented classification**. The proposed data-driven solution updates the state-of-the-art in thruster failure categorization in AUVs (Ranganathan et al. 2001). The approach presented in this work implements a Long Short-Term Memory Network (LSTM). In addition this solution is compared against other Machine Learning (ML) baselines. Data observation and analysis supports the design of a shallow NN by identifying a single sensor output for thruster fault classification.
2. **Task-oriented classification**. In the lightly explored topic of trajectory classification for autonomous vehicles, the proposed method for navigational trajectory categorization for mission monitoring establishes the state-of the art with NNs architectures. This study emphasizes the development of shallow architectures with rigorous hyper-parameter tuning experiments when dealing with an unbalanced dataset.
3. **Mobility hazard forecast and detection.** In the field of risk assessment for space rovers this work provides a valuable addition to the state-of-the-art with a data-driven approach implementing a shallow NN for mobility hazard detection and prediction. Compared to other recent approaches which combine multi-modal data or rely heavily on visual cues that provide some knowledge of the world. This work uses an unsupervised learning approach to analyze Inertial Measurement Unit (IMU) sensor input and sense imminent risks in mobility without any previous knowledge of the terrain.

- **Hyper-parameter importance for model selection.** In the field of Explainable Artificial Intelligence (XAI), Shapley Additive Explanations (SHAP) are generally used to explain a models feature learning. In this work is used in a different context to assist in the selection of candidate models suited for deployment. In addition, this application explains the performance of the model based on the tuned hyper-parameters in order to boost the trustworthiness of the selected model given its shallow configuration.

1.5 Document Structure

Separate chapters are dedicated to review relevant works in this field. The second chapter of this work identifies the research opportunities for different specific fields of field robotics based on the current state-of-the-art in autonomy. Particular attention is devoted to Anomaly Perception as it is a fundamental component of autonomy. Works are organized for Underwater and for space and ground robots.

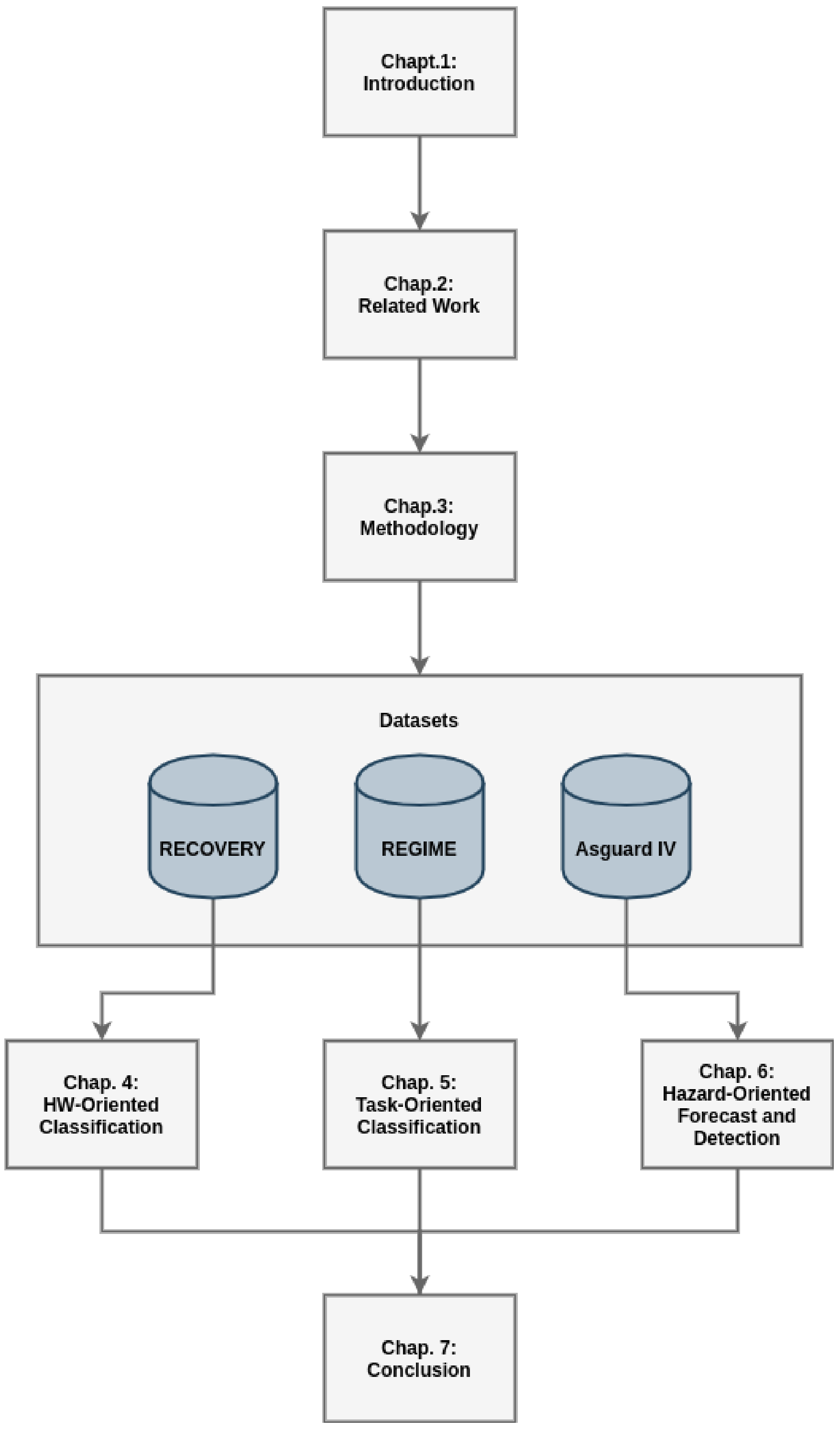

Chapt.1: Introduction
Chap.2: Related Work
Chap.3: Methodology
Datasets
RECOVERY
REGIME
Asguard IV
Chap. 4: HW-Oriented Classification
Chap. 5: Task-Oriented Classification
Chap. 6: Hazard-Oriented Forecast and Detection
Chap. 7: Conclusion

Figure 1.2: Document structure and association.

The third chapter revises the fundamental theory and methodology that is implemented throughout this work, focusing primarily in data handling practices and evaluation techniques that are common in ML applications. In addition, the fundamentals of the different NN architectures that have been implemented in this work are also discussed.

The fourth, fifth and sixth chapters address the described contributions. As shown in figure 1.2, which illustrates a graphical flow of the content, each chapter will present a problem described by a dataset. The RECOVERY and REGIME datasets are used for supervised learning problems and thus, they are both labelled. These datasets describe situations common in AUVs. The AsguardIV dataset is used to implement methods as an unsupervised learning problem and hence, it is unlabelled. This dataset describes events incurred in mission executed by a space rover.

The seventh and final chapter gives a summary of contributions. This chapter is also dedicated to discuss learned lessons and future work.

1.6 Publications

As first author:

1. De Lucas Alvarez, M., & Lane, D. M. (2016). *A Hidden Markov Model application with Gaussian Mixture emissions for fault detection and diagnosis on a simulated AUV platform*. OCEANS 2016 MTS/IEEE Monterey,, 1–4.

2. De Lucas Alvarez, M., Hastie, H., & Lane, D. (2017). *Navigation-Based learning for survey trajectory classification in autonomous underwater vehicles*. IEEE International Workshop on Machine Learning for Signal Processing, (MLSP), 1–6.

3. De Lucas Alvarez, M. & Kirchner, F. (2021). *Hazardous Mobility Forecast and Detection in Space Rovers*. Unpublished. Submitted to journal of Field Robotics.

As coauthor:

1. Vögele, T., Wehbe, B., Nascimento, S., Kirchner, F., Ferreira, F., Ferri, G., Machado, D., Phillips, A. B., Salavasidis, G., & De Lucas Alvarez, M. (2016, June 3). *ROBOCADEMY - A European Initial Training Network for underwater robotics. OCEANS 2016 - Shanghai.*

2. Valdenegro-Toro, M., De Lucas Alvarez, M., Dmitrieva, M., Wehbe, B., Salavasidis, G., Heshmati-Alamdari, S., Fuentes-Pérez, J. F., Yordanova, V., Istenič, K., & Guerneve, T. (2019). *Results from the Robocademy ITN: Autonomy, Disturbance Rejection and Perception for Advanced Marine Robotics*

Chapter 2

Related Work

”In the beginning, there was man. And for a time, it was good...
Then man made the machine in his own likeness.”
-The Instructor, *The Animatrix*

This Chapter presents the current state of the art on various topics relevant in this dissertation. The works reviewed here fall within two topics. The first is the line of works that are the groundwork for NN performance for anomaly detection across various disciplines. This will help establish that it is one of the most common and sought applications in autonomous robots. The second is the design and implementation of NNs in relevant topics to autonomy enhancement for Robotics and Autonomous Systems (RAS) in field operations. The two specific topics worked in this dissertation are addressed: trajectory classification and anomaly perception.

The next section addressing the first topic will review the advancements of NNs are highly relevant to this dissertation which have not necessarily have been implemented for RAS. Since the resurgence of NNs (Hinton 1988), the advancement in technology has enabled a leap in performance in diverse ML applications (Bishop 1995) and evolved the field into DL (Schmidhuber 2015). These works will establish the fundamentals and the performance to which a shallow NN should align. A brief introduction to the formalization of autonomy as it pertains to RAS is first given.

These works not only present the diversity of challenging situations which autonomous vehicles have to confront, but also show the incredible importance these are for industrial and societal infrastructures (Wong et al. 2017). The applications are widely broad. This is why this Chapter considers only the relevant topics and applications that align to the problems presented in this dissertation.

2.1 System Monitoring with Neural Networks

The works reviewed in this section discuss NNs approaches to System Monitoring (SM) in general. These mostly encompass fault detection and accommodation as the previous sections. However, these are not for applications concerning autonomous robots.

Early applications of neural networks into power plants showed the utility of these methods for fault detection and diagnosis (Xu and Wesley Hines 1999; Simani et al. 1999; Gaura and Kraft 2002). Continuous sensor monitoring and calibration provided useful information to operators to achieve unnecessary maintenance and reduce equipment damage ultimately providing efficient operation of the power plants. Standard Radial Basis Function Network (RBFN) and Multi-Layer Perceptron (MLP) implementations are used.

Solutions for fault detection in actuator systems such as motors or other types of rotating machinery have also been implemented in Jack and Nandi 2002 and Selmic et al. 2006. More specific applications within this topic such as aircraft systems have also been approached in Innocenti and Napolitano 2002. Fault detection in lower level components such as sensors was also studied in Zhang, Bingham, et al. 2012. The authors used a combined approach of Self-Organizing Map (SOM) neural networks for observing the operation of industrial turbine systems monitored with an array of sensors.

The incorporation of temporal analysis in fault detection becomes relevant as the complexity of the problems arise. The work of Jomeiri 2010 addresses the problem with time-series and thus they employ Recurrent Neural Networks (RNNs). NNs have also been implemented on the monitoring of software and network traffic. An interesting notion from this work is that it establishes that a shallow RNN architecture performs well fault identification problems. The work presented in Obst 2013 proposes a variant of RNNs called Echo State Networks (ESNs) which are efficient in computational complexity. They combine this method to create a spatio-temporal model to diagnose and detect faults in a sensor array. They define their frameworks as Spatially Organized Distributed (SONs) to learn correlations between different sensors that could be malfunctioning.

Time-characterization for sensor behaviour was also approached in Jager et al. 2014. The authors use a Temporal-Delay Neural Network (TDNN) to detect different sensor faults using different limit checking methods for the different fault classes. Their work presents the implementation of a single neural network to detect four different fault types. An interesting notion from this work is that, the network is quite successful at identifying two distinct faulty behaviours. However, the remaining faults were not recognized by the TDNN. This poses the question of whether the delay component in the framing of the problem could be causing this behaviour. Instead of using the sensor inputs, the authors use feature extraction based on mean, standard deviation, deviation, a correlation coefficient and signal-to-noise ratio. For similar signal behaviours these features could consequently yield similar feature scores, which could become complex for a NN to discriminate.

The work by Hussain et al. 2015 proposed another time-invariant approach with a NN. The authors suggest a Fully Connected Cascade Neural Network (FCCNN) for fault detection and identification of manually injected sensor faults applied to an aircraft sensor suite. The sensor readings correspond to the rotational forces roll, pitch, yaw. Their approach to design their networks was to take each of these sensor readings and construct

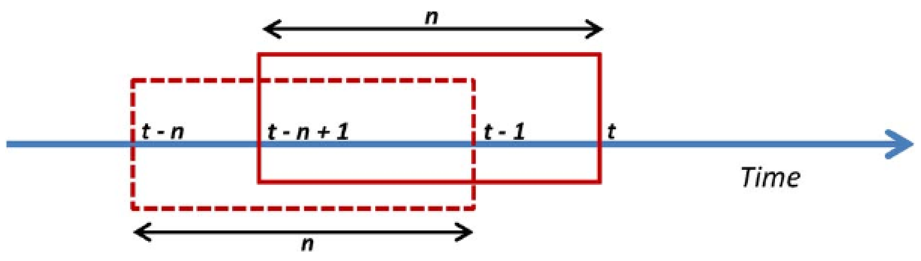


Figure 2.1: Sliding window at time *t* from Hussain et al. 2015

a FCCNN for each. The evaluation of their frameworks consisted in reporting the average detection time for each of the sensors. In comparison to the previous work, these features are used as is. The fault categories are similar to the previous work as well and the authors incorporate a delay in the form of a residual generated by a sliding window (Fig. 2.1).

By this time, works were starting to present implementations with LSTMs. In Juba et al. 2015 the authors implement Stacked LSTMs for anomaly detection in a number of types of data: ECG, space shuttle, power demand and multi-sensor engine. Their approach focuses in modeling nominal behaviour to detect deviations without defined context or processing. The authors compare their results against a standard RNN and evaluate them with standard accuracy and prediction metrics. Their results show that LSTMs outperform RNNs but still have margin for improvement in the prediction metrics and depth.

LSTMs continue to be used for anomaly detection. The work of Medel and Savakis 2016 proposed Predictive Convolutional LSTM for anomaly detection in videos. This is one of the works that introduce the combination of LSTMs with other NNs for processing various types of information. The authors also subsume their architectures under Encoder-Decoder frameworks split for present and future prediction. The authors evaluated the anomaly detection using error metrics and show that their Encoder-Decoder performs better than the Autoencoder model.

An interesting work by Cheng, Xu, et al. 2016 proposes a Multi-Scale LSTM for Border Gateway Protocol traffic anomaly detection. They refer to Multi-Scale at their handling of Internet traffic as a multi-dimensional time sequence. The authors empirically establish lengths of time-windows and define their goal to match the high-performance obtained from their literature review which is 99.5% accuracy and attempt to reduce the false rate. Their data is labeled and preprocessed and thus framed as a classification problem. They conclude their study by presenting close to their desired results, however they do no specify the size or length of their network.

In the work by Erfani et al. 2016 the matter of scalability and computational efficiency is addressed for solutions that integrate deep learning architectures. The authors propose a hybrid solution that combines a Deep Belief Network (DBN) and a one-class Support Vector Machine (SVM) trained with the features extracted form high-dimensional data by the DBN. Their novel framework for anomaly detection shows that expensive non-linear kernel machines can be replaced by linear when combined with DBNs. Their results reach the same accuracy as stat-of-the-art Autoencoder while considerably lowering its training and testing time.

It is worth adding three more works to reinforce the versatility and flexibility of LSTMs. Although NNs have been used to medical problems and data for at least 20 years, the work by Lipton et al. 2016 is the first to address this in a medical application context by implementing these networks for clinical diagnosis. The authors address data processing with target replication by sliding time window and the regularization of RNNs establishing that Dropout is only applied to the non-recurrent weights of the network. Their three-layer networks show promising results that could benefit from more rigorous hyper-parameter tuning.

In Hsu 2017 the authors model temporal correlations by combining RNNs and variational inference. Their approach is applicable to the inspection of monitoring data from various domains that handle a variety of high-dimensional data from different domains. The work by Lu et al. 2017 addresses the problem of anomaly detection in image processing using a variety of datasets from different fields ranging from military to education. They focus on the issue of the effect noisy data can have on the performance of the models. They implement an Autoencoder to capture the difference between anomalies and nominal data and integrate that with RNNs for detection. The author approaches the training aspect using a layer-wise procedure instead of maximum-likelihood estimation to simplify the process and achieve scalable training. The authors present their results with standard accuracy metrics as the ground truth and labels are available. However, they are presented as mean metrics since they represent the evaluations across cross-validation and therefore omitted model selection and re-training with the full data.

2.2 Autonomy Enhancement in RAS

Whether it is land, underwater or space missions, field operations are increasingly becoming more challenging due to harsh environments. This wide range of applications for RAS and their related environments have clearly established the urgent need to eliminate the presence of human interaction (Endsley and Kaber 1999; Stubbs et al. 2007; Beer et al. 2014; Wong et al. 2017). There is a consensus among these works that the study of Human Robot Interaction (HRI) is required to reach certain levels of autonomy in robots. These works agree in organizing autonomy into generally four tasks:

1. Monitoring for information acquisition.
2. Information analysis.
3. Decision selection.
4. Action implementation.

In terms of robot autonomy these are condensed into four main actions:

1. Autonomous information acquisition and data transformation
2. Autonomous information analysis and interpretation.
3. Autonomous decision selection.
4. Autonomous action implementation.

Past and current works commonly integrate one or more of these actions. Furthermore, the scope of autonomy actions can be bounded into different categories according to the functions of autonomous robots. In field operations, robots are usually task-performing vehicles that execute navigation, perception and management functions. It is common for research and applications to cover more than one of such functions. The following section reviews works which have served as inspiration and motivation to this dissertation and they encompass the four principal autonomy actions into relevant problems and applications.

2.3 Trajectory Classification

Trajectory classification for mobile robots is a lightly explored research problem. This is probably due to the lack of a critical underlying motivation. A section is uniquely dedicated to this topic as it does not fit into the anomaly perception to field but it is relevant to the discussion of task-oriented context categorization.

An early work (Sas et al. 2003) addressed the uses of navigational trajectories within a virtual environment. The principal goal was to identify behaviours, procedures and strategies that rule the way in which humans decide their movement paths. Interestingly, the authors propose a clustering Artificial Neural Network (ANN) that uses visual data with the focus of anomaly detection. While the primary interest in this study supports the development of surveillance methods, it establishes the importance of replacing human operators through autonomous systems.

The problem of trajectory classification on moving vehicles using ML appears soon after starting with the work by García et al. 2006. In the topic of air traffic control, the authors classify segments of four different modes of flight: uniform motion, transversal, longitudinal and combined maneuver. The ML method is based on a decision tree algorithm that generates partial rules for classification (Quinlan 1993). The performances of this classifier is compared to a traditional Interactive Multiple Model (IMM) tracking filter that processes the data in a backward and forward manner over the data and then uses an edge-detection algorithm that is plainly outperformed by the ML method.

More works also demonstrate the relevance of trajectory classification using diverse data and approaches for various applications with moving agents. The authors in Lee, Han, Li, et al. 2008 approach a feature extraction method that segments trajectories and subsequently generates two classifiers. The first, a high-level-feature cluster that categorizes trajectories based in regions not taking into account movement patterns. The second, a low-level based cluster that classifies trajectories based in navigational pattern.

Another work (Panagiotakis et al. 2009) integrates the temporal component to model similarities among the trajectory segments and subsequently grouping similar line segments into clusters. This work integrates methods and knowledge from Giannotti et al. 2007; Lee, Han, and Whang 2007 and Lee, Han, Li, et al. 2008 where the usage of time is being used as an informative feature for this application. Their experiments show how their approach discovers sub-trajectories from real trajectory data.

Later works present a series of methodologies that extract features and patterns to classify trajectories. However, these do not always focus on applications to moving vehicles. The classes range from types of human movements (Li 2014) or moving object (Biljecki 2012). While classifying movements a similar trend in the methods from trajectory categories are present. Such are traditional ML methods like Decision Trees Zheng, Chen, et al. 2010; Zheng, Liu, et al. 2008 and NNs in Byon et al. 2009. SVMs have also been implemented in Dodge et al. 2009.

Hidden Markov Models (HMMs) have been popular addressing the classification of trajectories. The work presented in Bashir and Khokhar 2007 compares the performance of their main method, HMM, against Principal Component Analysis (PCA) extraction of features coupled with a Gaussian Mixture Model (GMM) for trajectory density features. The authors in Mlích and Chmelar 2008 continue along the lines of security surveillance by implementing a human path classifier in an underground station. Reddy et al. 2008 presented a prototype classification system that consisted of a decision tree followed by a HMM.

Some works extended the HMM to more elaborated frameworks. For instance, the work presented in Liao, Patterson, et al. 2006 and Liao, Fox, et al. 2007 used Hierarchical HMMs to model movements of people by hierarchically dividing the activities into segments into a graphical temporal representation. Nascimento et al. 2010 also extended to Switched Dynamical HMMs for classification of pedestrian trajectories.

A digressing field of research Activity Learning could be said that has taken inspiration from of the latter works as the popular HMM has also been used for trajectory learning for robot activities. An example of such is the work by Osman et al. 2017 where the authors use such a HMM for processing demonstrated trajectories into generalized trajectories for robot learning. Along the same topic, the work of Lee and Ryoo 2017 now uses Convolutional Neural Networks (CNNs) for predicting locations of human hands and objects to plan the trajectories of a robot manipulator.

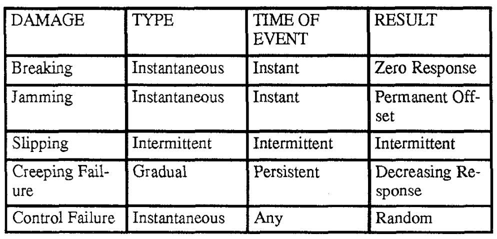
It is clear that trajectory classification has been mostly popular with topics of surveillance and few are connected to the implementation of autonomous vehicles. The narrow research in this problem opens the door to improvements in performance with NN as it is evident how it can provide enhanced autonomy to unmanned robots.

2.4 Anomaly Perception

Any perceived or anticipated action and event is subject to unforeseen factors. This is due to a variety of aspects ranging from missing information, to non-deterministic events inherent to the system or environment. This motivates the highly explored topic of anomaly perception. This encompasses detection, isolation, anticipation and mitigation. This is a well studied topic with thousands of works produced from various scopes and applications. In the interest of bounding the relevant literature, this section reviews works specifically dedicated to robotics in the fields of water, land and space.

2.4.1 Underwater Robots

An early work by Rae and Dunn 1994 introduced the concept of Intelligent Damage Detection in AUVs based on existing literature at the time that documented failed AUV missions due to simple system failures. The authors fundamentally addressed the problem through level monitoring and failure categorization.



DAMAGE	TYPE	TIME OF EVENT	RESULT
Breaking	Instantaneous	Instant	Zero Response
Jamming	Instantaneous	Instant	Permanent Offset
Slipping	Intermittent	Intermittent	Intermittent
Creeping Failure	Gradual	Persistent	Decreasing Response
Control Failure	Instantaneous	Any	Random

Table 1. : Damage Types

(a) Types of faults

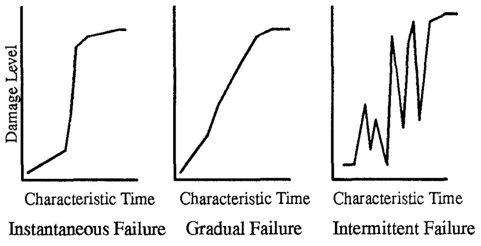


Fig. 1. : Types of Damage

(b) Fault-Time characterization

Figure 2.2: Damage Types from Rae and Dunn 1994

This was meant to address the some issues that the solutions at that time were lacking, which was finding the source and extent of the problem where the response of a system controller is merely reactive. By monitoring the vehicle at various levels, they were able to implement various signal gradient filters, each representing a category of the failure (Figure 2.2), they were able to provide multi-level coverage in the system under an Auto Regressive model. This approach used an linear approximation of the dynamics of the vehicle.

Subsequent works started to directly incorporate dynamic models for the identification of actuator faults in AUVs. The authors in Alessandri et al. 1998; Alessandri et al. 1999; Bono et al. 1999 used a Proportional–Integral–Derivative (PID) controller based in an open-loop surge thrust selection and a closed-loop steering controller. A bank of estimators is used to calculate a fault hypothesis and this is then measured against residuals to detect discrepancies in the predicted behaviour.

Detection of faulty behaviour was then complemented with fault accommodation known as Fault Diagnosis and Accomodation System (FDAS). The work presented in

Podder and Sarkar 2001, introduced a novel approach to the allocation of thruster in redundant configuration to compensate for faults. This approach takes into account thruster redundancy in terms of Cartesian space to resolve the compensation solution. Their simulations results are presented in terms of viability.

A series of works followed where system fault detection, with particular focus for thruster malfunction was approached by implementing sensing frameworks. The work by Hamilton et al. 2001 proposed a concept of heterogeneous knowledge for fault diagnosis. The concept of heterogeneous knowledge refers to the definition of knowledge spaces for the detection and diagnosis of faults. Concretely, they defined three inter-related spaces: fault, observation and diagnosis space. The observation space is characterized by the number of sensors embedded in the vehicle. The fault and diagnoses space are sets of possible outcomes that can occur on the vehicle that use the observation space. This approach leans more into a semantic solution for determining the source of faults in a system referred as Model-Based Diagnosis not to be confused with previous works that integrated dynamical models.

The work by Omerdic and Roberts 2004 integrates both, in which the thruster dynamics is necessary for the allocation process in the Fault Accommodation subsystem. A semantic relationship is also established between the thruster fault states, types and remedial actions similar to the work of Rae and Dunn 1994. Both the work by Omerdic and Roberts 2004 and Hamilton et al. 2001 have embedded the Fault Detection system in the vehicle.

The work by Ranganathan et al. 2001 proposes an intelligent system for failure detection and control in AUVs. This work is relevant as they are interested in mission viability in the event of minor failures in the sensors and control issues. Their proposed method integrates a two-layer ANN with 16 input nodes and 32 hidden nodes for nine classes of faults and a fuzzy rule-based expert system for inferencing the failures by observing the various changing parameters on the dynamics of the vehicle.

The increasing complexity of FDAS led to more comprehensive monitoring of AUVs. However, not all solutions could be embedded in the vehicle. The work of Wang, Wu, et al. 2008 designed a broad monitoring system for an underwater vehicle that in addition to sensing the dynamics controller, it also spanned the navigation and vision systems both optical and acoustical. The fault categories were bounded to three categories that characterized the behaviour of the signal instead of a semantic fault. Wavelet processing was used for this purpose. The fault allocation is managed by threshold-based rules.

A solution presented in Wang, Zhao, et al. 2008 used RBFNs that managed the fault compensation through signal restoration. RBFNs generated similar signal outputs to compensate the controller faults. Similarly, Zhang, Wu, et al. 2009 performed simulation experiments with RBFNs for thruster fault compensation. An alternative approach by Corradini et al. 2011 proposed the reduction the of observed features from the controller model for thruster failure compensation.

A series of works began to emphasize the relevance of time and managed their input as time-series or sequences. The work by Bian et al. 2009 implemented a Grey Model. Fundamentally, this model describes the dynamic characteristic of the input by finding the mathematical relations and law between factors based on the characteristic information. Their method generates sequences based on the sensor data and generates residuals for the parameters in the model to then predict if the sequence corresponds to a fault.

The author in Qin and Gu 2009 proposed an on-line method based on a Gray Model to diagnose the sensor faults for AUVs. Their model describes the behaviour of an input data sequence and predictions are derived and compared with the current measurements to generate a prediction error sequence on four typical sensor fault modes.

This solution is extended by the Jia et al. 2013a; Jia et al. 2013b where they reiterate the use of the Grey Model. The authors employed a Grey Correlation Analysis to determine the similarity of the geometry factor change curve to decide the relevance between the factors. Thus comparing the correlation degrees between the sequences to determine residuals. The authors later incorporated a Second-order Taylor series dynamic prediction to compensate for sparse and incomplete data. Chen et al. 2010 also implemented a method similar to these, also consisting of signal construction, but rather instead of a Grey Model, the authors used a Strong Tracking Filter (STF) to generate a predicted sequence. The solutions based in sequence prediction for residual calculation all present results in simulations.

Graphical diagnosis model approaches have also been proposed. The work of Dearden and Ernits 2013 suggested an automated fault diagnosis system for an AUV using a Bayesian Network for estimating risk of vehicle loss in an under-ice mission. The network was built the AUVs control nodes architecture and historical observed failure occurrences. The authors in Raanan et al. 2016 also propose automated fault diagnosis in an underwater vehicle using on-line topic models, which is a variant of a Bayesian non-parametric model, the Latent Dirichlet Allocation (LDA). Commonly used to analyse texts, LDA automatically infers the number of classes in the data. The authors used this method to automatically characterize patterns in the vertical plane performance of the vehicle.

The work of Fagogenis et al. 2016 implemented a solution for modeling an occurring thruster fault. A Bayesian filter to detect deviations from the normal operations of a thruster system in an AUV. The authors use a nominal and an adaptive model, which is intended to capture the motion of a vehicle after a fault as a Mixture of Gaussians distribution. The divergences between these two models is capture determines whether the vehicle is compensating for a fault without knowing the dynamics of the vehicle. The suggested this approach is meant to improve nominal or adaptive control.

More recent work has incorporated a solution with a similar goal. In Wehbe and Krell 2017; Wehbe, Hildebrandt, et al. 2017 the author proposed a solution for motion model identification using the thruster rotational features. In this case, the dynamics of the model was incorporated and after a series of standard ML model comparisons they concluded that kernel-based non-linear estimators outperformed NNs and Least-Squares estimators. In Wehbe, Arriaga, et al. 2018, the author later extends his work to identify multiple contexts of an AUV. In this instance, a LSTM network is implemented which outperforms standard ML.

2.4.2 Space and Ground Robots

Autonomous robots for ground applications often have human experts monitoring the mission as generally it is often possible for humans to be in close proximity. However, there are situations in which this may dangerous or just inaccessible to humans. Hence it is obvious that anomaly detection methods are necessary for these applications.

The literature reviewed in this section addresses mostly fault detection in the localization and navigation systems. Authors in Monteriù et al. 2007b defined a method for determining signal residuals of wheel encoders, IMU and Global Positioning System (GPS) sensors. In a following publication (Monteriù et al. 2007a) the authors were able to test the proposed system on a mobile robot platform. Their results showed that the sensor faults were detected, isolated and reported to the ground control station in all situations where a single sensor fault. In the event of multiple sensor failures the detection was possible but isolation was not always guaranteed.

Model-based reasoning has been often implemented in systems designed for space applications (Ferrell et al. 2010; Bozzano et al. 2011). These are often on-board systems that highly complex models that monitor high and low level subsystems that generate and monitor mission plans. Such model-based approaches have also been discussed in the previous section for underwater vehicles and therefore is not unusual to see similar approaches across domains.

An early work (Inotsume et al. 2013) presented an analysis and evaluation of the effects of rover reconfiguration on the its traveling performance on a lateral slope of loose soil. The authors presented a model that represents the relationships among rover attitude, contact forces, and slippage thus providing an understanding of rover behaviour on loose soil that would lead to more effective mobility control strategies. While their approach only considered lateral traversal, the authors recognize the need to extend their method to uphill and downhill dynamics.

In the study by Köhler et al. 2013 evaluated different ML-based solutions for sensor fault detection and compensation in autonomous space rovers. They compare two different methods, a Neural Gas and a MLP for sensor sequence prediction. The evaluation of fault types is modeled by considering sensor value behaviour. Similar to the work of Rae and Dunn 1994 the authors consider characterization based on temporal behaviour and synthesize three types of fault classes. The platform used was a four-wheel skid-steered rover. Four sensor modalities taken from the IMU are used as input. The inclusion of optical-flow supplied by a camera was integrated in two separate evaluations for both models. The predictions from these are then compared to the sensor channels for categorization based on absolute errors. The compensation is managed by a separate module that based on the error model, the vehicle is supplied with the expected sensor readings generated by the prediction. Despite both models being successful, the authors were not conclusive in determining which method was better.

Some interesting works in the field of hazardous event classification are relevant to mention. In the work by Bouguelia et al. 2017 the authors introduce an unsupervised method for classifying by clustering of a space rover slip events based on proprioceptive sensors. The authors use a traditional method, Bayesian tracking, for updating and improving the parameters of the models as new input data comes in. The authors report their method outperforming a K-means solution with a score of 86% vs 80% accuracies. The success of this works establishes that it is possible to design and implement unsupervised models to detect unforeseen and unstructured events.

The following works introduce the importance of monitoring traction in planetary exploration rovers as it can impact the mission by negatively impact navigation or immobilizing a vehicle. The work by Gonzalez, Apostolopoulos, et al. 2018 also addresses the issue of slippage. Their approach consists in evaluating two supervised machine learning methods, SVMs and NNs, against two unsupervised learning approaches, K-Means and SOM. Their physical experiments were performed on a single wheel testbed with IMU sensors placed in a chassis location where the detection would be maximized. The authors results show that a SOM-based algorithm balances the advantages of supervised and unsupervised learning algorithms which are high success rate and low storage requirements respectively.

The authors later extend their work in Gonzalez, Chandler, et al. 2019 by performing rigorous tuning to evaluate their algorithms with 55 configurations. They also incorporate other types of settings that they consider can influence the performance of the models. Such are rover speed, type of terrain and type of tire. In a different publication (Kruger et al. 2019) they increase settings evaluation like a sandy incline, more slip classes, different rover velocities and sensor inputs. Their SOM solution still outperforms compared to other algorithms.

An interesting work by Skonieczny et al. 2019 presents a data-driven approach combining exteroceptive and proprioceptive inputs to measures mobility risk in a space rover. This assessment categorizes the rover interaction with the surface into three risk categories, thus assisting in the early detection of potentially hazardous changes in terrain conditions.

More recent works address the problem of terrain identification for survey rovers. In Banerjee et al. 2020 the authors developed a framework that models wheel-terrain dynamics in a rover that is able to adapt rapidly to the change of terrain. The authors have done so by capturing non-linear interactions in the dynamics of the controller and the terrain. The linear model is augmented with new data learned with features learned from a NN with a Bayesian regression algorithm.

The same problem is approached in Dimastrogiovanni et al. 2020 where a comprehensive set of proprioceptive sensors are used to classify terrain. Hazardous events are subsumed under the classification as slippage is treated as a type of terrain that can be detected by the classifier. The authors implement a SVM using class dataset they have gathered from their own robot. The authors report robust results.

2.5 Summary

Monitoring for autonomous robots envelops many application studies for the enhanced supervision of autonomous robots while they are carrying out tasks. It is clear from this research that classification and prediction of sequences of sensory data for behaviour detection are intrinsic components of perception that support an autonomous vehicle's functions. Collectively these works have all addressed the issue of behaviour or fault detection some of which incorporate event accommodation.

The contribution these works have set in regards to fault detection are valuable to the field. It has been shown that the usage of dynamical model is applicable but not required to achieve good results. With regards to failure detection, the works have shown that often there are specific faults that are of interest to model and that often it is in the thruster system in which these applications are implemented.

Fault accommodation is also an important goal in some of these works, although they are naturally a consequence of the identification of a malfunction and of systems that carry redundant hardware or control systems capable of adapting to such events. Few works address the matter of embedding the solutions in the vehicle. Thus, there exists an opportunity for improving the deployability of equally robust or better implementations.

A common characteristic of these monitoring systems is that they are built to be overseen by human experts, either on-line or off-line and few works approach this from a embedding perspective. Another research opportunity that has been identified is that almost none of these works discuss rigorous hyper-parameter tuning specially when implementing NNs. This is due to the *de facto* usage of non-embedded processing units, therefore there is no interest in optimizing the size of the model.

Regarding unsupervised methods for classification and detection, the data used in almost all of the reviewed works is separated by classes therefore is possible to use evaluation metrics used in supervised learning. These are important points that are addressed throughout this dissertation.

Chapter 3

Methodology

"I'm super brain. That's how they made me."
-Goldfrapp, Lyrics to *Utopia*

This chapter reviews the Neural Network fundamental theory and methods that are used throughout this dissertation. First, standard ML procedures and practices are addressed as they are the starting point to the three problems discussed in this work. Such are the various ways of evaluation, data handling and structuring which are relevant for sequential and time series data. As with most machine learning problems. Also, three different strategies of hyper-parameter tuning approaches were used and therefore discussed.

Afterwards, the fundamentals are discussed. The main methodology implemented in this dissertation has been a type of NN specifically designed to process time series, LSTMs. These networks have been pervasively implemented and successful at solving problems with sequential data. Other support networks like CNNs and Dense Networks are also employed and described in this Chapter as well. Combinations of LSTMs with other frameworks have also been a approached. In this case the Encoder-Decoder and Autoencoder structures.

3.1 ML Standard Practices

3.1.1 Evaluation Criteria

There are many standard criteria for evaluating the robustness of a ML algorithm (Goodfellow et al. 2016). However, these metrics are not meant to be used indiscriminately. There are two main groups of evaluation criteria which are suited either for classification or regression assessment.

Firstly, some terminology needs to be clarified to approach classification metrics. In the topic of predictive analytics, accuracy can be deconstructed as a table reporting number of counts of *hits* and *misses* per class. These are denoted as True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) to describe the count of correct *hits*, correct rejections, false alarms, and *misses* respectively. This table, also known as confusion matrix, will give a more detailed understanding of accuracy, particularly if a dataset is unbalanced.

Accuracy is defined as the percentage of correctly classified instances by a model and is expressed as,

$$Acc = \frac{TP+TN}{TP+FP+TN+FN} \tag{3.1}$$

This metric is appropriate when True Positives and True Negatives counts are of more interest. In the case of unbalanced datasets, there are two relevant metrics that are useful for describing retrieval of relevant instances. One of which is Precision, also known as Predictive Positive Value (PPV), and is defined as,

$$Precision = \frac{TP}{TP+FP} \tag{3.2}$$

This describes the proportion of retrieved positives that were classified correctly. Recall, also known as sensitivity or True Positive Rate (TPR), is defined as the harmonic mean of precision and recall,

$$Recall = \frac{TP}{TP+FN}. \tag{3.3}$$

This describes the proportion from real positives that were actually classified correctly. Unbalanced class distribution exists in most real-life classification problems. Hence, using the aforementioned measures, it is possible to evaluate the accuracy of a model on an unbalanced dataset. The F1 score is used when the counts of FN and FP are most important. The F1 score is defined as,

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}. \tag{3.4}$$

In regression, these metrics are not useful, as the problem requires the evaluation of a ML model's output approximation to a specific functional behaviour. For this, the R^2 score can be used to measure the goodness of fit. It is defined as,

$$R^2 = 1 - \frac{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N}(y_i - \bar{y})^2} = 1 - \left(\frac{SSE}{SST}\right) \tag{3.5}$$

where N is the total number of observations, SSE is the sum of squared error and SST is the total sum of squares. The R^2 score is good for evaluating the prediction of a continuous variable and is the fraction of response variance that is captured by a specific model. If $R^2 = 1$, this means that the model fits the data perfectly. Hence, a high R^2 is desired.

The metrics discussed in this section will be the default assessment methodologies that will be used in the problems addressed in this dissertation.

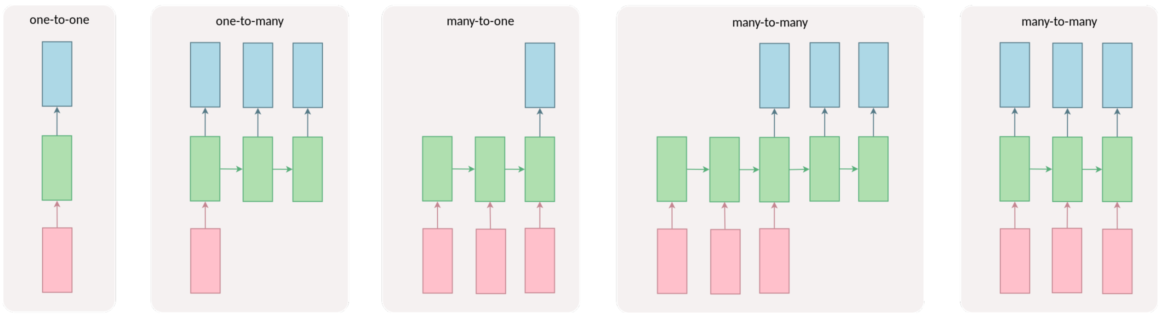


Figure 3.1: Sequence classification input-output. The rectangles represent vectors and the arrows are the matrix multiplication operations in the recurrent neural network. Input are red, output vectors are blue and green vectors are the RNNs state. (one-to-one) Standard mode of processing from fixed-sized input to fixed-sized output, e.g. single sequence to single class. (one-to-many) Single sequence classified into multiple outputs or have a sequence output. (many-to-one) Multiple sequence input classified into a single class. (many-to-many) After a number of multiple sequence inputs, there is an output of multiple sequences or class. (many-to-many) The synchronized case.

3.1.2 Problem Framing for Sequences and Time-series

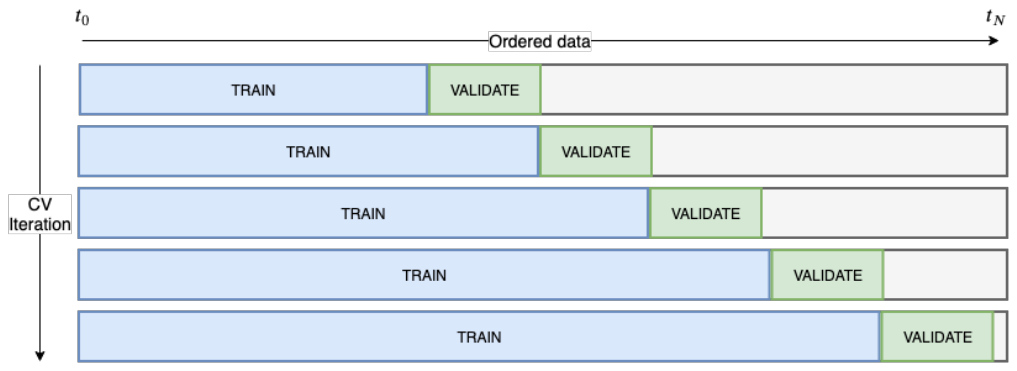
While it is unmistakable that the use of LSTMs implies working with data expressed in consecutive time, the manner of framing supervised and unsupervised problems requires some sequence or sample engineering. One of the advantages of using LSTMs is that, when modeling sequences, the inputs and outputs can be adjusted according to the requirements. Some problems may require for us to use sequences as inputs, outputs or both. Figure 3.1 shows concrete examples of how these are formatted.

Sequence input and output organization is important for information extraction. For this work and in the context of classification, the method applied is many-to-one using the sliding window method. In the context of forecasting, the selected method is many-to-many using the time series fold method.

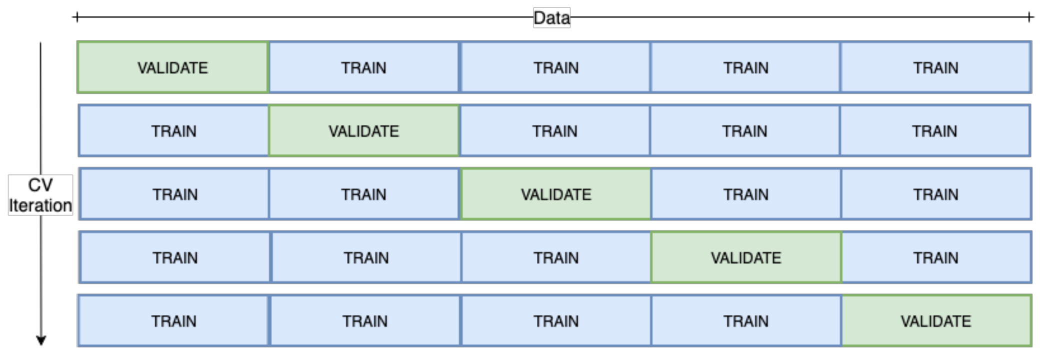
3.1.3 Cross-validating Time-series

The applications explored in this dissertation involve implementing ML methods with time series. The datasets used in this work come from sensor readings and the chronological order in which they come is to be preserved for context extraction. When implementing ML algorithms, it is recommended that cross-validation is used during the training process. However, cross-validation can become problematic for pattern detection in ordered data, such as time-series. Traditional cross-validation splits a set of data into n subsets or folds taking away one subset for validation and using the rest for training.

This is not something that can be done for sequences. For this there is Forward Chaining or Time-series Fold. These processes, unlike in standard cross-validation or K-Fold, training sets are super-sets of those that come before them in order to maintain the sequential ordering successive. Figure 3.2 illustrates a comparison of standard



(a) Time-series fold



(b) K-Fold

Figure 3.2: Cross-validation methods.

cross-validation and time-series fold.

Time-series cross-validation (Fig 3.2a) is usually implemented for prediction problems while K-Fold is used for classification. In this case the full sequence needs to belong to a single class for the validation folds to work.

Because in this work, the full sequences include more that one class, the data is first processed into class segments. In the case of sequence prediction, instead of a time-series fold, the data is folded into samples using a sliding-window to fit a supervised learning problem. This is done to address two issues:

1. Preserving the same sample size, even though LSTMs are well able to do this at the expense of excessive padding.

2. Increasing the amount of samples, as the datasets are already small.

Similarly for classification, this fold can also be used saving on each iteration one of the windowed segments for validation. This is illustrated in Figure 3.3. Consider having a sequence of length t, which is then split into a set of N pairs $(X, \hat{X})$ that corresponds to *train* and *validation* sets for classification, or in the case of prediction, into the $S_{t_{0-w} \to t_0}$ input and $S_{t_0 \to t_{0+w}}$ with one step overlap.

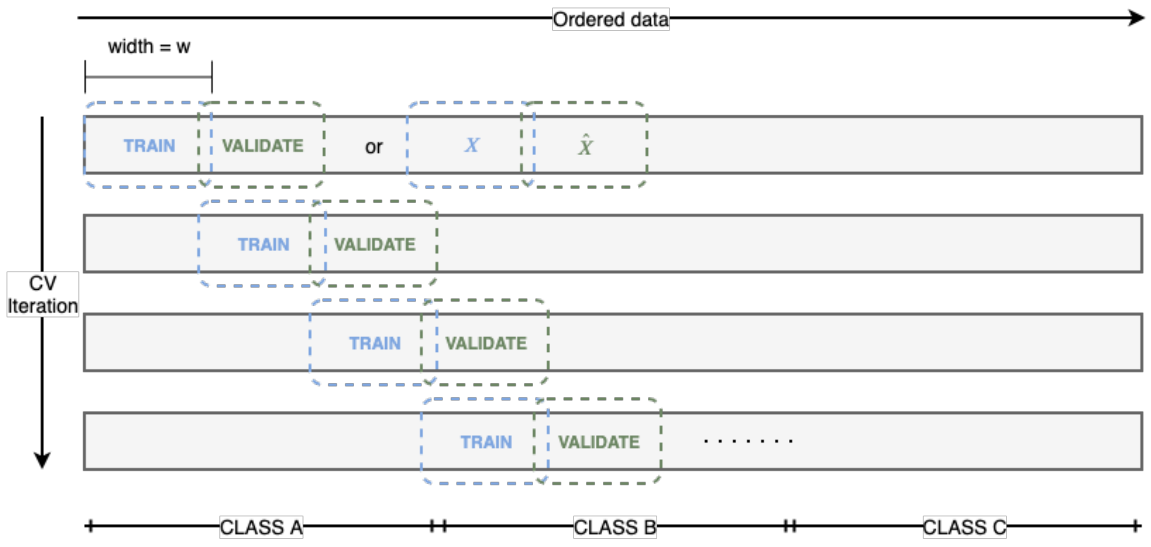


Figure 3.3: The solid grey block represents the full sequence, S_F. If this is a classification problem, it is segmented into subsets that represent categories maintaining chronological order. It is assumed that three categories if different context occur consecutively once, $S_F =< S_A, S_B, S_C >$. A sliding window of size w is run through each set generating a set of *train* and *validation* pairs if its classification or $(X, \hat{X})$ if it is a prediction problem.

3.1.4 Hyper-parameter Tuning and Regularization

Hyper-parameter Search. This work implements several methodologies for hyper-parameterizing LSTMs. In NNs, a parameter refers to typically the weight of a node which is determined through the training process. In contrast, a hyper-parameter is used to control the training process and are not derived through it, rather they affect the speed and quality of the training process. Hyper-parameters would be considered the elements that define the topology of the NN such as number of nodes and depth, or learning specifications such as learning rate and batch size.

The most exhaustive of the hyper-parameter search approaches is the Grid Search. This is because this search will output every possible combination given a range of values to set for each element in a hyper-parameter set, λ. This can become very costly in terms of computing power and time, specially if cross-validation is performed.‘’

On the other hand, Random Search has shown to be more efficient than the latter even though it is not as costly. In Random Search, given a range of values, a random set of combinations is yielded. In Bergstra and Bengio 2012 the authors show empirically and theoretically how randomly chosen hyper-parameter sets are more efficient for optimization than Grid Search.

Another method for hyper-parametrization used in this work is Hyperband Bayesian Optimization (HBBO) (Falkner et al. 2018) in which Bayesian Optimization (Shahriari et al. 2016) is combined with Hyperband (Li et al. 2018) to optimize progressively faster than Random Search and Bayesian Optimization and Hyperband alone. Bayesian Optimization (BO) models the objective function as, $p(f|D)$ based on the already seen data points, $D =$

$(x_0, y_0), ..., (x_{i-1}, y_{i-1})$. An acquisition function $a = X \to \mathbb{R}$ based on the current model compensates between exploration and exploitation. Algorithm 1 shows this process.

```
Algorithm 1: Bayesian Optimization
  Input: $x \in X$
  Select the point that maximizes the retrieval function.
  $x_{new} = \arg\max_{x \in X} r(x)$
  Evaluate the objective function.
  $y_{new} = f(x_{new}) + \varepsilon$
  Augment the data and refit.
  $D \leftarrow D \cup (x_{new}, y_{new})$
```

Hyperband (HB) is a hyper-parameter optimization strategy that uses Successive Halving (SH) (Jamieson and Talwalkar 2016) to identify the best n randomly sampled configurations. This is achieved by evaluating approximate versions of the objective function with a limited budget, $\tilde{f}(\cdot, b)$ of $f(\cdot)$ with a budget $b \in [b_{min}, b_{max}]$. Successive Halving evaluates the chosen configuration on the given budget and ranks them based on their performance. It then takes the top η^{-1} on a η times larger budget. Algorithm 2 shows this process.

```
Algorithm 2: Hyperband
  Input: $l$: budgets $b_{min}$ and $b_{max}, \eta$
  Computes the geometrically spaced budget $\in [b_{min}, b_{max}]$
  $s_{max} = [\log_n \frac{b_{min}}{b_{max}}]$
  Repeat until the maximum budget is reached
  for $s \in \{s_{max}, s_{max} - 1, ..., 0\}$ do
      sample $n = [\frac{s_{max}+1}{s+1} \cdot \eta^s]$ configurations
      run SH with $\eta^s \cdot b_{max}$ as initial budget
  end
```

HB has shown that it can outperform random search and Bayesian optimization, however it is limited when it comes to converging to the global optimum due to the randomly chosen configurations even when large budgets are chosen. For this matter, HBBO integrates HB and Bayesian Optimization through the Tree Parzen Estimator (TPE) as it is simpler than standard Gaussian Process (GP)-BO as they tend to require complex approximations. The TPE is Bayesian Optimization method (Bergstra, Bardenet, et al. 1994) that uses a kernel density estimator to model the densities over the input configuration space rather than modeling the objective function f directly by $p(f|D)$. The algorithm produces a variety of densities over the configuration space X using different observations $x^{(1)}, ..., x(k)$ in the non-parametric densities. TPE defines $p(x|y)$ with two densities,

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{3.6}$$

where $l(x)$ is the density formed by the observations $x^{(i)}$ such that resulting loss $f(x^{(i)})$ is less than y^* and $g(x)$ is the density formed by the remaining observations. By not using all observations HBBO is more computationally efficient for HB to carry many functions evaluations at small budgets.

Concretely, HBBO chooses budgets as in HB including SH, but instead uses BO. The Bayesian component of HBBO closely resembles TPE with the difference in that the density estimation is done for a single multidimensional Kernel Density Estimator (KDE). Algorithm 3 shows the sampling process.

Algorithm 3: Hyberband Bayesian Sampling

```
Input: : observations $D$
Input: : fractions of random runs $\rho$
Input: : percentile $q$
Input: : number of samples $N_s$
Input: : minimum number of points $N_{min}$ to build model
Input: : bandwith factor $b_w$
Output: : next configuration to evaluate
if $rand() < \rho$ then
    return random configuration
end
$b = \arg\max\{Db : |Db| \geq N_{min} + 2\ \}$
if $b = \emptyset$ then
    return random configuration
end
Fit KDEs according to 3.6 and 3.7
Draw $N_s$ samples according to $l'(x)$
return sample with highest ratio $l(x)/g(x)$
```

First, to maintain the theoretical guarantees of HBBO, a constant fraction ρ of the configurations is uniformly sampled at random. This ensures that after $m \cdot (s_{max} + 1)$ runs (SH), on average $\rho \cdot m \cdot (s_{max} + 1)$ random configurations have been evaluated on the maximum budget, b_{max}. Since the goal is to optimize on the largest budget, HBBO always uses the largest budget for which the largest number of observations are available. This $N_b = |D_b|$, with budget b, is large enough to satisfy $q \cdot N_b \geq N_{min}$.

To calculate the KDEs a minimum number of data points, N_{min}, are required satisfying the condition $d + 1$, where d is the number of hyper-parameters. However, instead of waiting for $N_b = |D_b|$ to be satisfied, $N_{min} + 2$ configurations are initialized and the best and worst configurations are used to model the two densities:

$$\begin{aligned} N_{b,l} &= \max(N_{min}, q \cdot N_b) \\ N_{b,g} &= \max(N_{min}, N_b - N_{b,l}) \end{aligned} \tag{3.7}$$

Finally, Expected Improvement (EI) (Jones 2001) is optimized by sampling N_s points from $l'(x)$, i.e. the approximation KDE of $l(x)$ but with rather all bandwidths multiplied by a factor of b_w to support exploration on more promising configurations.

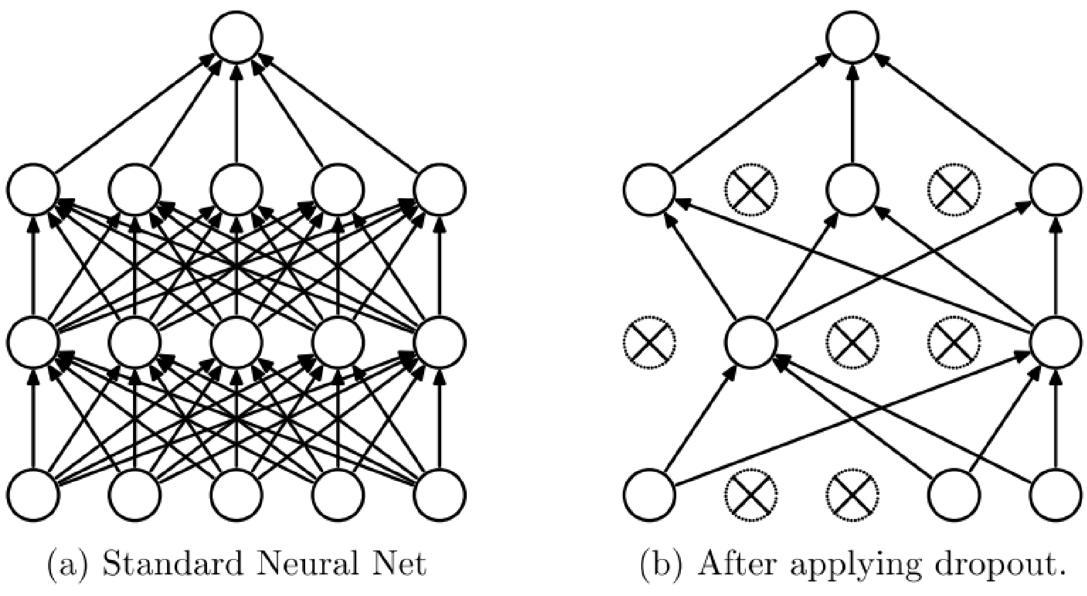


Figure 3.4: From Srivastava et al. 2014. a) A standard fully connected network with no dropout. b) The same network with dropout.

Dropout. When working with neural networks, it is common that networks that are too large for the available dataset can over-fit the training data. Dropout is a regularization criterion that helps the model generalize better (Hinton et al. 2012; Srivastava et al. 2014). For each sample, some layer units from the network have a likelihood of being omitted, or dropped out, by a specified factor.

This has the effect of temporarily ’removing’, or dropping out, a specified percentage of nodes randomly from the networks in each update. This in consequence causes the networks to have a different layout of incoming and outgoing connections from the previous update. Figure 3.4 shows how this works for a fully connected network. The principle works the same way for LSTMs. This way each layer result is not bound or dependent to the outputs of previous layers. This layer dependency, although beneficial, can sometimes tend to over-correct the errors from previous units and cause over-fitting and non-generalization to unseen data.

An additional advantage of using Dropout, is that it induces sparse representations. This is caused by the dropout effect simulating a sparse activation in a layer. This is beneficial where there is a need for models that yield sparse representations such as Autoencoder models.

Dropout is implemented per layer in the network. However, it can be implemented on any layer of the networks except the output layer. Because the dropout factor is a likelihood, an additional hyper-parameter must be included in the tuning process which specifies the probability at which the nodes are kept or discarded. Dropout has also the effect of pruning the networks as it trains. For this reason it is recommended that the size of the network is increased, i.e. increasing the number of nodes.

Having reviewed two important regularizing techniques relevant for this application, the set of LSTM architectures have therefore been evaluated using Dropout in the LSTMs and Class Balancing. The next section will detail the range of hyper-parameter values used in the training process.

3.2 Hyper-parameter Importance

While neural networks have demonstrably provided advanced solutions in real-world applications, they are still considered a black-box solution by some and their tuning is considered an art to some experts. Existing methodologies for hyper-parameter optimization can be time-consuming. Methods like Grid-Search and Random-Search for finding optimal hyper-parameter values in neural networks can become inefficient and stagnate fast development of neural networks. Faster methods like Bayesian optimization has been shown to require less iterations than Random-Search and still achieve better performance on the held out test set (Bergstra, Yamins, et al. 2013).

These techniques are important to know as it helps in the implementation of robust networks effectively. The question of why it is that these chosen hyper-parameters perform the better than the rest of candidates is still explored by many researchers. The abstraction of explanations in ML is done to clarify and interpret the predictions of a model by relating the feature values of an input sample to its output prediction in a way that a human expert can understand. These methods or algorithms for explainability harness valuable properties (Robnik-Šikonja and Bohanec 2018) that add trustworthiness to a model not just by its accuracy metrics but by the transparency and reliability they provide when deploying a ML learning model into the real world.

There is a selection of methods dedicated to analyse specific models and others that are model-agnostic (Ribeiro et al. 2016). One relevant methodology is the finding of Shapley values for the input features (Shapley 1953). A method inspired by game theory, its general idea is to consider each feature as a 'player' in a game and the prediction as the reward. In game theory terminology, the Shapley value could be defined as the distribution of the reward among the features. In ML the Shapley value is a method for calculating the contributions of each input feature on single predictions for any ML model.

3.2.1 Shapley Value

The Shapley value of a single feature is the contribution to the reward and its calculated as the average of the marginal contributions across all permutations,

$$\phi_j(val) = \sum_{S \subseteq \{x_1,\ldots,x_p\}/\{x_j\}} \frac{|S|!(p-|S|-1)!}{p!}(val(S \cup \{x_j\}) - val(S)) \tag{3.8}$$

where S is a subset of the input features used to train the model, x is the vector of output values of the reward, i.e. the instance to be explained and p the number of total features. To get the overall contribution of specific coalitions in a subset, i.e. the prediction of feature values in S marginalized over features that are not included in S is,

$$val_x(S) = \int \hat{f}(x_1,\ldots,x_p)d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X)) \tag{3.9}$$

this is done for each feature not contained in S. All possible combinations of feature values need to be evaluated with and without the j-th feature to calculate the exact Shapley value. This requires considerable computing time. The authors in Štrumbelj and Kononenko 2014 proposed an approximation with Monte-Carlo sampling as the more features there are the more the possible combinations increases and the exact solution becomes problematic. Author Molnar 2019 summarizes it so,

$$\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m)) \tag{3.10}$$

where $\hat{f}(x_{+j}^m)$ is the prediction for the vector of feature values in this case of z, which is the random data point representing x, the feature values of the instance.

This still has to be repeated for each of the features to get all Shapley values. This also means that it cannot be used if sparse explanations are required, i.e. only just a few features, because Shapley explanations require all the features.

3.2.2 Shapley Additive Explanations (SHAP)

SHAP (Lundberg, Allen, and Lee 2017; Sundararajan and Najmi 2020) is a method derived from Shapley value and other methods on model interpretability explanations that can provide explanations for individual predictions with a few features. The SHAP method also computes the Shapley values for explaining how the reward, or in this case prediction, is distributed among the features. The innovation that SHAP introduces is that the Shapley value is presented as an additive feature attribution method. SHAP defines the explanation as,

$$g(z') = \phi_0 + \sum_{j=1}^{M}\phi_j z'_j \tag{3.11}$$

where g is the explanation model, $z' \in \{0,1\}^M$ is the coalition or feature vector, M is the maximum size of the vector and $\phi_j \in \mathbb{R}$ is the Shapley value for a feature j. Three important feature of SHAP for model interpretability are:

Algorithm 4: Approximate Shapley Estimation

```
Input: $N$: Number of iterations
Input: $x$: Instance of interest.
Input: $j$: Feature index.
Input: $X$: Data matrix.
Input: $f$: Machine learning model.
Output: $\hat{\phi}_j$: Shapley value of the j-th feature.
for $m$ =1,...,$M$ do
    Draw random instance z from the data matrix X
    Choose random permutation o of the feature values
    Order instance x:
    $x_o = (z_1, ..., x_j, ..., x_p)$
    Order instance z:
    $z_o = (z_1, ..., z_j, ..., z_p)$
    Construct two new instances, with feature j:
    $x_{+j} = (x_1, ..., x_{j-1}, x_j, z_{j+1}, ..., z_p)$
    and without feature j:
    $x_{-j} = (x_1, ..., x_{j-1}, x_j, z_{j+1}, ..., z_p)$
    Compute marginal contribution:
    $\phi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j})$
end
Compute Shapley value as the average:
$\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m$
```

1. **Global Interpretability.** This refers to the collective SHAP values contributing to the prediction. This shows the negative of positive influence of each feature on the prediction.

2. **Local Interpretability.** This focuses in giving each observation or individual instance its own set of SHAP values to explain why each individual case contributes in a specific way to the prediction.

3. **Model Agnostic.** SHAP can explain the output of any machine learning model using a Kernel explainer. It can also explain Tree-based modes, DL models and Linear models.

The novelty in this dissertation involves the usage of SHAP for understanding the hyper-parametrization of the models for model selection. The motivation is to explain why the best candidates perform better given their hyper-parameter configuration and use this to understand why these yield specific metrics. This is also expected to support the selection of a model that will generalize better.

This is done by formulating the hyper-parametrization as a regression problem, where the input features are based on the accuracy and scoring. The authors in Lundberg and Allen 2018 proposed a variant of SHAP, TreeSHAP, for tree-based machine learning models as a fast, model-specific alternative. TreeSHAP defines the value function as the conditional expectation instead of the marginal expectation to estimate the effects on the prediction,

$$E_{X_S|X_C}(f(x)|x_S) \tag{3.12}$$

It is important to mention that the explanations from Tree-SHAP, or any other ML for that matter, are not implying causality. Rather, by observing the correlations and patterns in the hyper-parameter values, they aid they support the model selection process by finding feature importance with feature effects and interaction among the SHAP values.

3.3 Learning on Neural Networks

While the used networks have different purpose and characteristics, their learning methodology can be generalized to all of these architectures: *Forward-Backward Propagation*.

As expressed in the name, Forward-Backward Propagation fundamentally consists of two processes, forward propagation and back-propagation (Rumelhart et al. 1986), with the goal of minimizing a gradient. The operations performed vary according to the respective network type, e.g. convolution in a CNN or gate operation in a LSTM. In addition, some functions within these two processes can also be tailored, i.e. the loss and optimization functions.

Forward propagation. In a way, the process of forward propagation has already been described for these architectures. It takes place with the target y propagating to the hidden units in each layer until output $\hat{y}$ is produced. Two important calculations occur here:Several computations take place here:

- $L(\hat{y}, y)$: Loss function depending on the output, $\hat{y}$, and target, y.
- $J = L(\hat{y}, y) + \lambda\Omega(\theta)$: The total cost, obtained with the output loss and added to a regularizer $\Omega(\theta)$.

Algorithm 5 lists how these values are calculated in forward propagation.

When training a neural network, this stage is repeated for a number of epochs which yields a total value from a cost function with respect to the parameters, $J(\theta)$ or loss. Fundamentally, forward propagation requires computing an affine transformation at each layer given the input from the previous layer and then apply an activation.

Back-propagation. Back-propagation takes the computed cost and propagates it backward through the network to calculate its gradient. This algorithm is conformed by two parts: the calculation of the gradient for the activations, a^k, in each layer k and the

Algorithm 5: Forward propagation

```
Input: $l$: Network depth
Input: $W^i, i \in \{1,...,l\}$: the weight matrices.
Input: $b^i, i \in \{1,...,l\}$: the bias parameters.
Input: $x$: the feature input.
Input: $y$: the target output.
The input feature vector x is the layer^0
$h^0 = x$
for $k$ = 1,...,$l$ do
    $a^k = W^k h^{k-1} + b^k$
    $h^k = f(a^k)$
end
$\hat{y} = h^l$
$J = L(\hat{y}, y) + \lambda\Omega(\theta)$
```

learning process that uses this gradient to update the weight values in the neural network with respect to the total calculated loss. The intuition behind these gradients is to asses how the outputs of each layer need to change to diminish the error. From these gradients, also the parameter gradient on each layer can be obtained.

Algorithm 6 lists how these gradients are calculated after forward propagation.

Algorithm 6: Back-propagation

```
$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$: output layer computation
for $k$=$l$,$l$-1,...,1 do
    Converting output gradient into pre-nonlinearity activation
    $g \leftarrow \nabla_{a^k} J = g \odot f'(a^k)$
    Computing gradients on weights and biases
    $\nabla_{b^k} J = g + \lambda \nabla_{b^k} \Omega(\theta)$
    $\nabla_{W^k} J = g h^{k-1\top} + \lambda \nabla_{W^k} \Omega(\theta)$
    Propagating gradients w.r.t the next lower-level hidden layer activation
     $g \leftarrow \nabla_{h^{k-1}} J = W^{k\top} g$
end
```

Figure 3.5 summarizes in a single image the learning process of a neural network. This is back-propagation in general terms and it can be applied to other tasks not just neural network learning. In practice, these methods require more complicated implementations. The naive version above, although straight-forward, only considers returning a single output, whereas generally it would be required to have more output. In addition, the memory management and consumption is also an important factor to take into account.

Modern implementations of this approach require the storing of data in tensors. A tensor is a container used to contain data in multiple dimension, e.g. storing matrices. In the simple approach, many tensors are computed separately and then added in a separate stage. This generates a high computational congestion which is solved in with a memory buffer that adds each value as it is calculated.

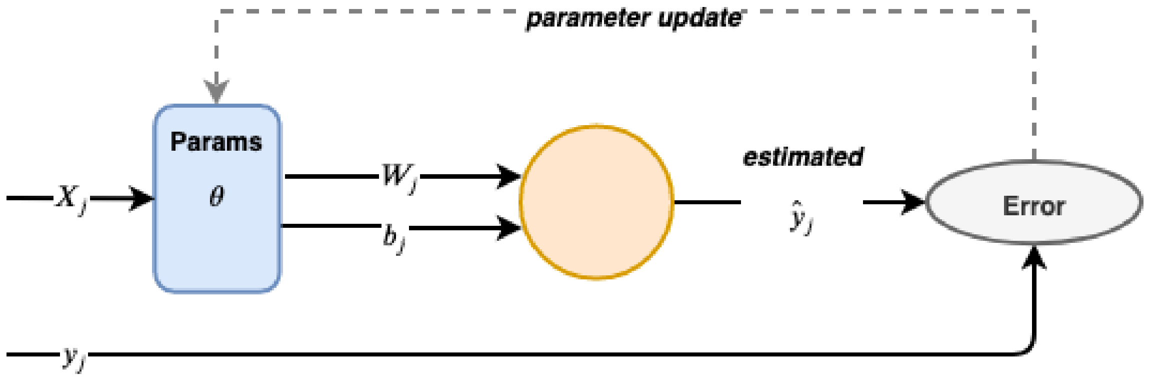


Figure 3.5: Visual summary of the learning process of one neuron in generalized stages.

NN-based solutions in field robotics require that these are embedded in the systems. Non-integrated processing overthrows the goal of real-time autonomous response as explained in previous chapters. It is for this reason that shallow and robust networks are required to provide autonomy enhancement to robots. The following section reviews the architectures that have been implemented throughout this work.

3.4 Examined Architectures

3.4.1 Long Short-Term Memory Networks

In the traditional RNN the learning methods that have a troublesome effect when multiplying operations to compute gradients. The aftermath of this multiplication is that the gradient can either decrease or increase exponentially. In Bengio, Frasconi, et al. 1994, it was demonstrated how this effect is problematic for capturing long-term dependencies by showing how the probability of training a robust RNN using Stochastic Gradient Descent (SGD) rapidly reaches 0 for sequences with length as short as 10. As discussed above, LSTMs have overcome this issue due to their internal mechanics summarize in these fundamental points:

1. LSTMs have three layers, or gates, that interact with the state in each cell. Sometimes, it is said there's four layers because the state is included. RNNs have just one.
2. These three interacting gates manage of information on the state: the *input*, *output*, and *forget* gates.
3. The state layer has a linear self-loop, in addition to the external recurrence of the RNN, whose weight is controlled by the *forget* gate. This is a key element for allowing the gradient to be maintained for longer.
4. These updates are performed in an additive manner, therefore the gradient practically never fades or burst.

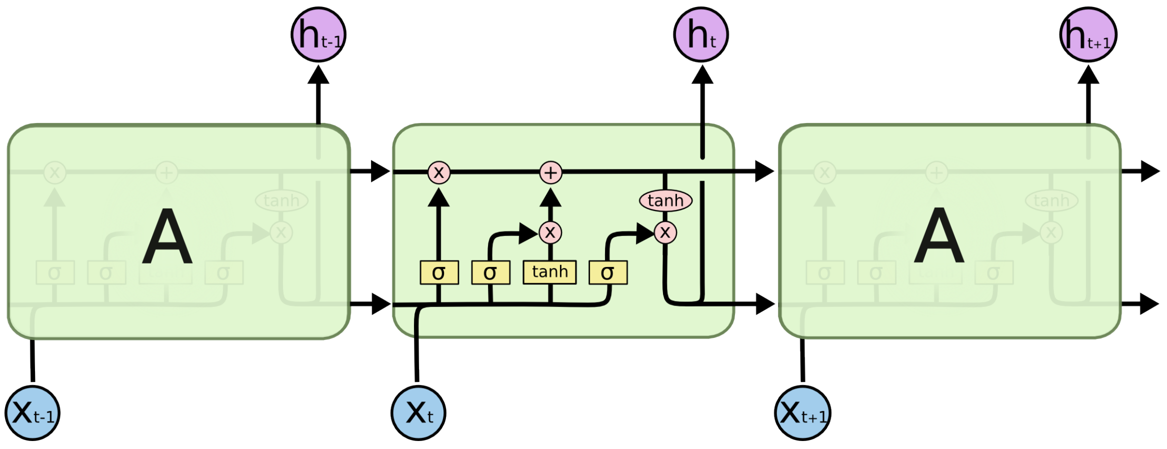


Figure 3.6: An unrolled LSTM cell with its four interacting layers. Credit: Olah 2015.

Figure 3.6 shows how the state C^t is computed by two point-wise operations from the info coming from the gates below. Similarly, each gate is also a point-wise operation with a sigmoid, σ, or a hyperbolic tangent function, tanh. The first operation decides if the state, C^t, is to disregard any information. This is controlled by the *forget* gate:

$$f_i^t = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{3.13}$$

where x^t is the current input vector and ht is the current hidden layer vector for time step t and cell or neuron i. The matrices b, U and W are biases, input weights and recurrent weights respectively for their corresponding gate as indicated by the superscript. Next, the decision to store new information is controlled by the *input* gate, g^t. This has two steps. First, a sigmoid operation filters the information that will be updated:

$$g_i^t = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right) \tag{3.14}$$

Before updating this to the state, a tanh layer that creates a candidate vector, $\tilde{C}$, updated to the state:

$$\tilde{C}_i^t = \tanh\left(b_i^C + \sum_j U_{i,j}^C x_j^{(t)} + \sum_j W_{i,j}^C h_j^{(t-1)}\right) \tag{3.15}$$

These two equations are then combined to update the old state C^{t-1} into new state C^t by:

$$C_i^t = f_i^t * C_i^{t-1} + g_i^t * \tilde{C}_i^t \tag{3.16}$$

Equation 3.16 multiplies the old state, C^{t-1}, with the output of the *forget* gate, f_i^t and adds the operation of the *input* gate. These are essentially values that have been scaled by the sigmoid and hyperbolic tangent operations to decide how much and with what the state value needs to be updated.

Finally, the *output* gate, o^t takes these preliminary values and filters them also in two steps. As before, the values are first filtered by a sigmoid operation.

$$o_i^t = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right) \tag{3.17}$$

Second, the cell state is passed through a hyperbolic tangent function to distribute the values an a range between $[-1,1]$. Unlike the sigmoid activation, the tanh operation allows for values to be positive and negative. This allows for state augmentation or reduction. This is then multiplied by the previous filtered result of the sigmoid gate and the cell output, h_i^t is generated:

$$h_i^t = o_i^t * \tanh(C_i^t) \tag{3.18}$$

Other variations of the standard, or commonly known *vanilla* LSTM, have emerged. The authors in Greff et al. 2017 provide in their paper a survey on these adaptations. They also offer a deeper introspection into the *vanilla* LSTM equations and its parametrization.

These variations essentially modify the internal connectivity of the cell by gate adaptation and/or activation function modification. The most prominent variations include the LSTM with peep-hole connections Gers and Schmidhuber 2000, which allow for all gates to look at the cell state, C^{t-1}, and the Gated Recurrent Unit (GRU) LSTM Cho et al. 2014. It has been suggested (Greff et al. 2017; Jozefowicz and Zaremba 2015), that there is no one variant that can surpass both the LSTM and GRU. In Goodfellow et al. 2016 it is summarized how some authors have investigated how the most critical component in the LSTM architecture is the *forget* gate (Greff et al. 2017) or how the LSTM can be boosted by adding a bias to this gate (Jozefowicz and Zaremba 2015).

The LSTM adaptations described above are not implemented in this dissertation. However, there are variations in terms of network architecture, where other types of neural networks are used in conjunction with LSTMs. Such are CNNs, the FCN, or Dense, and the more intricate Auto-Encoder. More detail on these variations is included in the Application Chapters 5 and 6).

3.4.2 Convolutional Neural Networks

The use of CNNs appears in the implementation of multiple sequence input (see many-to-one/many in Figure 3.1 from Section 3.1). CNNs are widely used for computer vision applications because they act in the same manner as biological vision systems, in which a group of cells, are tiled to cover the entire visual field and find spatial correlation from the input image.

Because CNNs are mostly used for image classification, they are generally used with 2-D data. However, there is a wide range of real-world applications in which the configurations of the CNNs can be 1-D (Kiranyaz et al. 2019) and 3-D. Hence, in

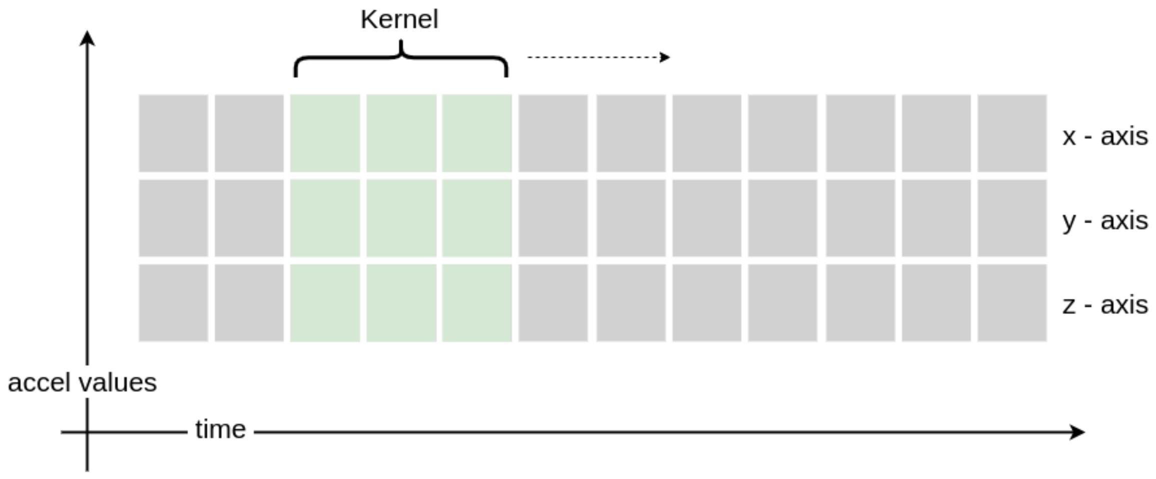


Figure 3.7: A filter with kernel size 3, striding along an accelerometer time series.

this instance the 1-D configuration is relevant for the applications on autonomous robots presented in this dissertation.

A time-series can be said to be 2-Dimensional, time being one dimension and the values of the data as the second. In this case, a one dimension convolution in a CNN works by sliding a filter along the time component. Hence, a CNN spatial extraction property works for time series data processing. Figure 3.7 illustrates this concept.

The main characteristics that distinguish CNNs from other MLPs are:

1. Unlike in other neural networks, CNNs share weights through the convolution operation.
2. This sharing of weights reduces the number of parameters and introduces sparsity.
3. Sparsity can be adjusted with the depth of the network, generating reduced feature vectors.
4. The reduction of the representation is functionally performed by max-pooling (or average pooling) where blocks of values are reduced to a single output.

The convolution operation is denoted as:

$$(I*K)_{i,j} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} K_{m,n} I_{i+m,j+n} + b, \tag{3.19}$$

where I is the input map or image, K the filter kernel of size $k1xk2$ and bias, b. Similarly for a 1-D convolution, the operation can be denoted as:

$$(I*K)_i = \sum_{m=0}^{k_1-1} K_m I_{i+m} + b \tag{3.20}$$

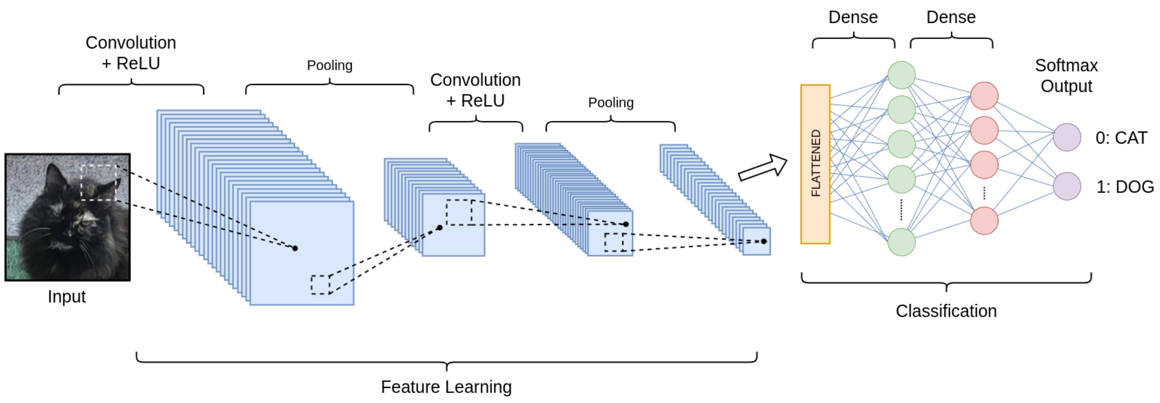


Figure 3.8: CNN network used to classify cat and dog images.

Sparsity. Generally, this term has a negative association due to its implications with information, i.e. sparsity in data would mean there is missing information. However, consider the case of a biological neural network. For every biological function, not all neurons are fired at once, only a select specific ones which are activated by different signals. This results in specialized and faster models that only process meaningful conditions of the problem to solve and with less parameters to train.

This convolution is combined with an activation function, usually a Rectified Linear Unit (ReLU) which is a non-saturating activation function

$$f(x) = \max(0, x) \tag{3.21}$$

thresholding at zero. This operation is performed between the convolution and the pooling layer. Before obtaining a result, the last features need to be flattened as they hold the shape of the pooling layer. A CNN us commonly combined with a FCN before the output activation.

Figure 3.8 illustrates how a common CNN architecture and contains all the elements and concepts just discussed.

For the goals described in this work, the time series used are multivariate, i.e. the sensors used in autonomous robots generally record information, such as gyroscopes and accelerometers record data in x, y and z axis. Figure 3.7 illustrates how a convolution kernel would move in an time series of accelerometer data.

In this manner, although LSTMs so far surpass any other model for learning patterns in sequential data, CNNs can improve the classifications of patterns by reducing the feature size that goes as input into a LSTM network. Figure 3.8 shows examples of CNN architectures for 3-D and 1-D input.

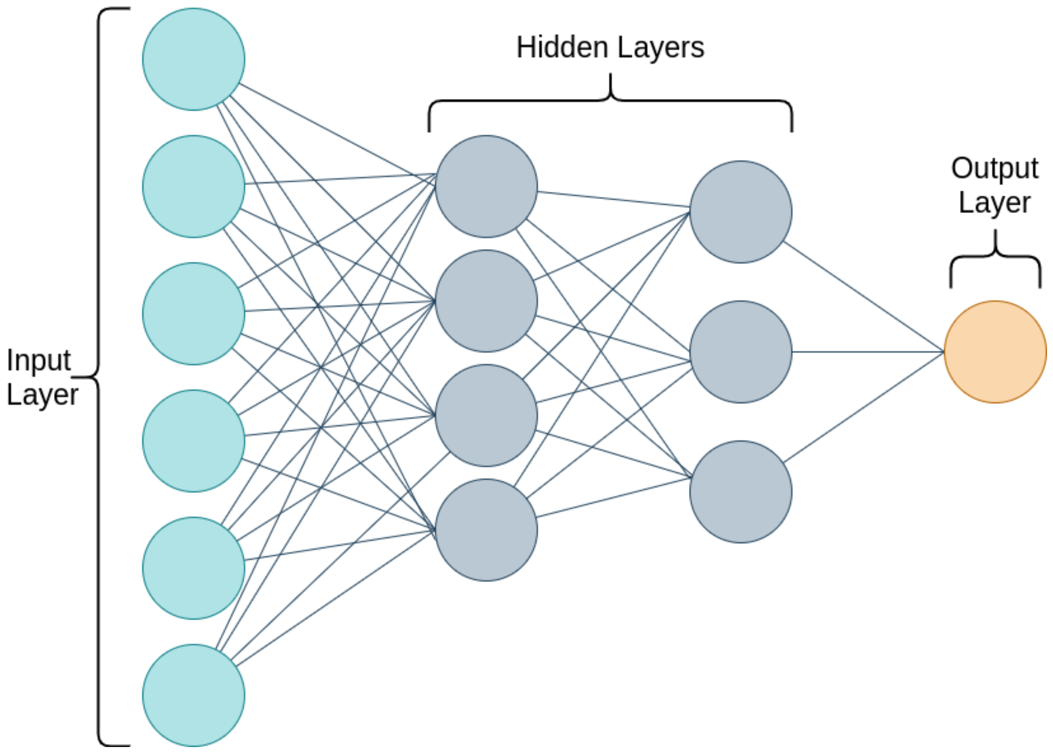


Figure 3.9: A representation of a FCN with one input layer, two hidden layers and a non linear output.

3.4.3 Fully Connected Network

A FCN is otherwise known in ML jargon as Dense Network and is the standard architecture for neural networks in which all neurons in one layer are connected to those in the next layer. While a Convolutional layer operates within a delimited field to collect persistent characteristics, this layer will obtain features from all the combinations of the features of the previous layer.

The features of a FCN are quite simple:

1. While a CNNs are only connected to neurons that are spatially close to them, neurons in a FCN will be connected to all neurons.
2. Like in a CNN, the FCN will also perform operations with activations functions and the weights before the output activation.

Figure 3.9 illustrates an example of a three-layer FCN. Because every input is connected to every output in a layer by weight, if the size and length of the network is large it can become computationally expensive having $Weights_{Total} = n_{inputs} * n_{outputs}$. However as support architectures, these are useful for flattening outputs from CNNs or LSTMs into final model outputs.

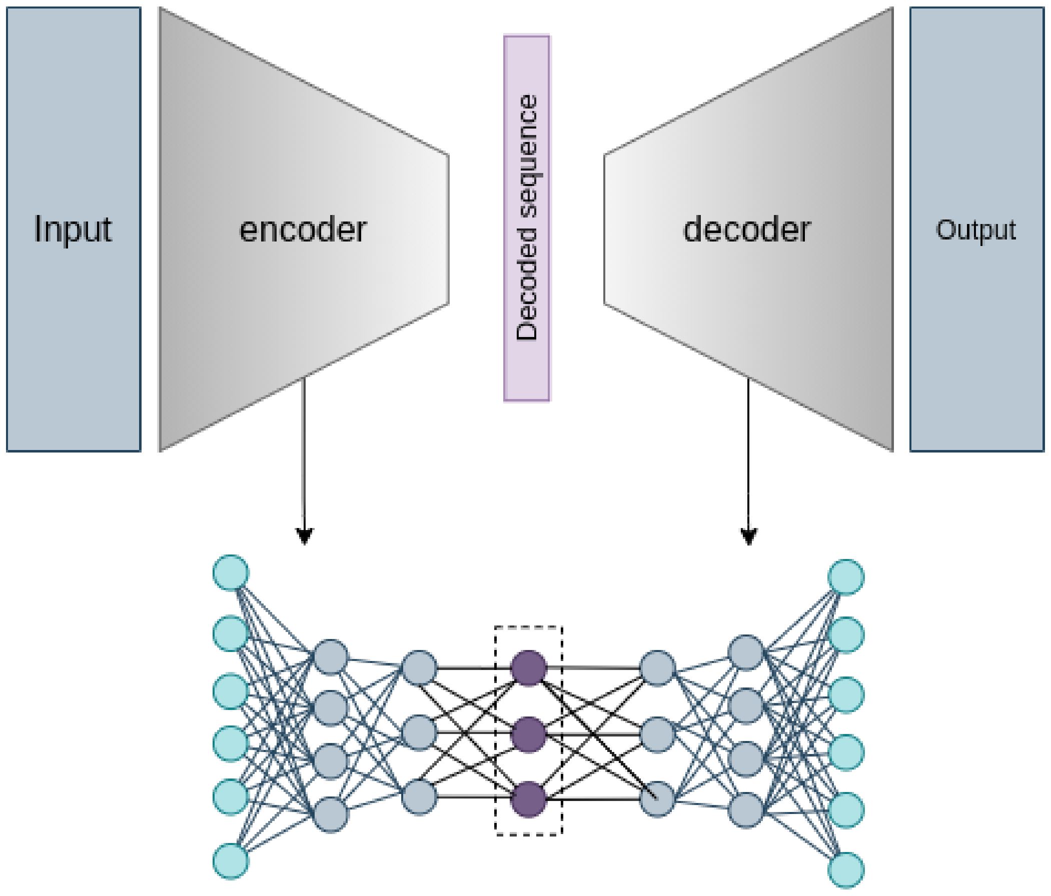


Figure 3.10: A NN Encoder -Decoder. An encoder produces a transformed representation of the input data formulating a latent space on which samples can be decoded into new examples.

3.4.4 Encoder-Decoder Framework

The Encoder-Decoder framework is a signal processing paradigm under which many models can be contained, including ML algorithms. This framework essentially has two components. The first is the encoder, a function that encodes an input signal into a latent or different space. The second is the the decoder, which takes the latent representation generated by the encoder and transforms, or decodes, it into an output.

This frameworks has been most pervasively used as sequence-to-sequence solutions in different applications. Any neural network can be used within an Encoder-Decoder framework to learn an optimized blueprint for encoding and decoding input. Figure 3.10 illustrates this concept. The most common application is from the field of Natural Language Processing (NLP) in machine translation problems (Sutskever et al. 2014; Cho et al. 2014).

For these applications, RNNs have been incorporated into Encoders-Decoders to yield time-distributed latent representations. Their success naturally sets precedent for the usage of LSTMs which have shown to be efficient learning time series, it has been explained in Section 3.4.1 that LSTMs overcome the vanishing-exploding gradient problem Goodfellow et al. 2016; Graves 2012. This is manifested when the propagation of the gradients through stack layers causes the gradient to become imperceptible or inflate so much that it becomes extremely difficult to adjust the weight appropriately causing the network to stop learning.

The application of an LSTM Encoder-Decoder in this work pertains to the problem of sequence forecasting. The LSTM Encoder-Decoder acts as a predictive model for input sequences so that it is possible to forecast sequences and anticipate events. The implementation of this architecture is further described in Chapter 6.

3.5 Summary

This Chapter consisted in presenting the fundamental methodologies that have been used throughout this dissertation. The first section goes through canonical practices when implementing solutions with ML. Though these are standard well-established procedures, they are fundamental processes that have been used in this work that provide strong support in the design and selection shallow NNs across the three different applications presented.

In addition, the theory and fundamentals of a method that determines feature importance in a model was also discussed. This method, Shapley Additive Explanations (SHAP) are used in this work in a novel direction that explains which hyper-parameters affect the robustness of the presented models.

Lastly, the theoretical foundations of NNs were explained. Particular focus was given to LSTMs networks, a variation of RNNs, that specializes in the processing of time series or sequential data. In addition, other frameworks and types of neural networks which can be used in conjunction with LSTMs were also reviewed as they were also implemented in this work.

Chapter 4

Hardware-Oriented Classification

"I collect information to use in my own way. All of that blends to create a mixture that forms me and gives rise to my conscience."
-Major Motoko Kusanagi, *Ghost in the Shell*

The categorization of hardware functionality is one elemental component of system monitoring as the condition of a hardware component can jeopardize the performance of the system. In the event of a malfunction, it is desired that such event does not go unidentified as it can result in fatal loss of an autonomous vehicle. While this notion of classification may seem apparent or trivial, it's important to observe and understand the data that determines these modalities. This problem can be approached as a supervised learning problem provided that the knowledge about the hardware function modalities are known. However, identifying malfunctioning signal patterns can be challenging in systems with multiple data sensors as values can potentially be mistaken as nominal or belong to more than one category.

Existing literature on the the topic of hardware monitoring has several contributions specifically in the field of anomaly detection for autonomous robots. Thruster malfunction identification is an application that is very popular in this topic. While many approaches have offered solutions for detection and classification, very rarely have they approached it from the DL perspective. In addition, thruster malfunction is rarely approached from a data driven perspective in which different types of hardware faults can cause thruster malfunction.

This Chapter presents a shallow NN-based solution to thruster malfunction classification. This is approached in a data driven manner where a number of sensors describe the nature of the thruster fault. The NN is based on a LSTM and is evaluated against established ML methods for time series analysis.

4.1 Application: Thruster System Failure

As part of system monitoring, classification of hardware component output supports the supervision and control of the hardware life-cycle or asset management. For AUVs that are deployed for extended periods of time, it is vital that hardware components can be monitored to protect the vehicle itself and the continuity of the mission. In this manner, the following data analysis and experiments model the output of a thruster system as a set of modes.

A formal representation of the data can provide insight into understanding the behaviours or context that arise.

Context Thrusters in AUVs are vital hardware components which are prone to failure or malfunction. These faults can be generated by environmental factors, e.g. stranded debris can obstruct the thruster, or induced by internal hardware components due to deterioration or end-of-life-cycle. These examples fall into both known and unknown categories. Firstly, because environmental factors are unanticipated and secondly, deterioration and decay of a component can be tracked. Therefore, it can become ambiguous at times to determine the causes of hardware malfunction, but highly relevant for fault classification.

Consider an AUV gliding at determined depths in open sea when it suddenly deviates from the specified course. Upon remote monitoring of the AUV experts determine that the cause for the path deviation is due to malfunction of a thruster determined by the anomalous voltage readings in the monitoring system. However, the exact nature of the fault requires further observation of the equipment as the state of the thruster system voltages alone cannot determine the exact cause. In addition, having this capability embedded as part of the AUVs Artificial Intelligence (AI) will circumvent the need of suspending missions explicitly for diagnosis.

This is a supervised classification problem in where the methods implemented are used to model hardware output that can be categorized into multiple classes. The following research additional questions are postulated for this Chapter:

1. How shallow can a Neural Network be to still outperform the classical ML methods?
2. Given the available different sensor outputs, i.e. features, in the dataset, which of these is best to use for this classification problem?

The RECOVERY dataset presents these characteristics for causes of thruster system failure. This dataset helps exemplify this clearly thanks to the fault-injecting capabilities of a thruster testbed used to collect data. The RECOVERY dataset provides four different sensor reading outputs: two motor voltages, current and thermal drift. While using all of these together could provide a rich representation of the class, statistical techniques

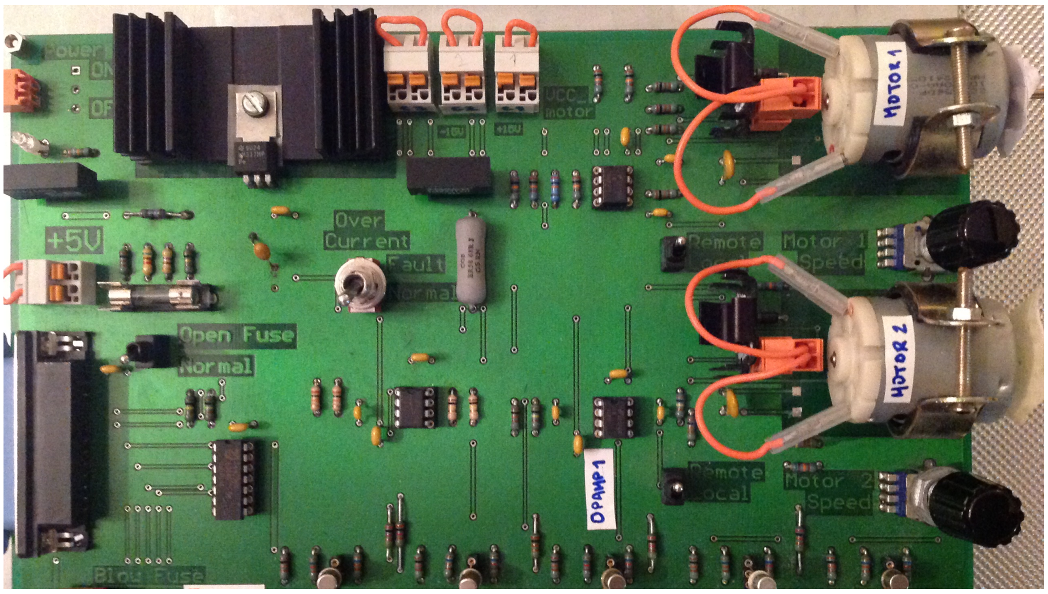


Figure 4.1: RECOVERY Testbed.

implemented in this work suggest that this may not be the case. The following section is dedicated to describe this dataset in more detail.

4.2 RECOVERY Test board Dataset

This dataset was collected to develop methods that diagnose faults which are intrinsic or caused by external irregularities at a hardware level. The values describe voltages, current and temperature measurements from a circuit testbed (Figure 4.1) originally designed for demonstrating aspects of a Fault Diagnosis software RECOVERY (Hamilton et al. 2001). The fault inducing capabilities of the testbed allowed to record nominal and four types of faulty operation:

1. **Cable 1 connection**. Manually detaching the cable from the terminal.
2. **Cable 2 connection**. Manually detaching the cable from the terminal.
3. **Thruster jamming**. Manually induced in the motor shafts.
4. **Over-current**. Activated by a switch that increases the current in a node leading to the motors.

The data was collected at approximately 10 Hz by manually inducing every class of fault and recorded the outputs without any preprocessing using a LabJack card. The data was also labeled in the same manner. All the features are numeric float values each representing voltages supplied, over-current detected and the thermal drift of the voltage supply in the circuit board, m_v^1, m_v^2, oc_v, d_v, respectively. There is a total of four features and there are no missing values. A preliminary statistical approach is used to understand the data and its discussed in the next section.

4.3 Data Observation and Analysis

A table was built to provide a qualitative assessment of component operation based on the specifications on numerical healthy operation values. Similar to the work of Rae and Dunn 1994 and Köhler et al. 2013, Table 4.1 shows an assessment on operation values per sensor in the testbed according to fault class. With this information a qualitative assessment of component operations defines the numerical operation states of the sensors and a hypothesis can be built for each to have distinct distribution attributes for normal and faulty variations of hardware performance.

Fault Class	m_v^1	m_v^2	c_v	d_v
Normal	nominal	nominal	nominal	nominal
Cable 1	nominal	nominal	degradation	nominal
Cable 2	surge	surge	instant degradation	intermittent nominal
Thruster Jams	nominal	nominal	intermittent surge	intermittent degradation
Over-current	intermittent surge	intermittent degradation	nominal	nominal

Table 4.1: Qualitative assessment of hardware component operation.

The variable values from each sensor can be visualized for each fault class in Figure 4.2. These visuals can help in identifying which features will be the most useful. From this graphical perspective it can be observed that the motor sensors, m_v^1 and m_v^2, show stable values without any outliers for the five categories of thruster system behaviours. The d_v and oc_v features show to be the most relevant for the fault cases. Each class behaviour is defined below for the current and drift sensor:

1. **Cable 1.** In this case, both features seem to reach their maximum and lowest values, respectively.

2. **Cable 2.** The behaviours for both features in this case appear abnormal as well, with the over-current, oc_v, completely flat lining as in the previous case. Thermal drift, d_v, shows an increase in its normal operating value ranges.

3. **Motor Jams.** For this fault category, both features show a complete display of abnormal behaviour where both sensor outputs jump out their normal ranges intermittently caused by the external source blocking the motors.

4. **Over-current.** Similarly to the motor jam fault, this class shows how the values fall out of their normal ranges.

A more exploratory analysis can aid in understanding the correlations between these features. Figure 4.3 shows pairwise relationships in the data. The diagonal axes show the histogram of the sensor in that column. Based on this study it is concluded that the data does not represent the simplicity of the problem as the value ranges and distributions can represent multiple categories.

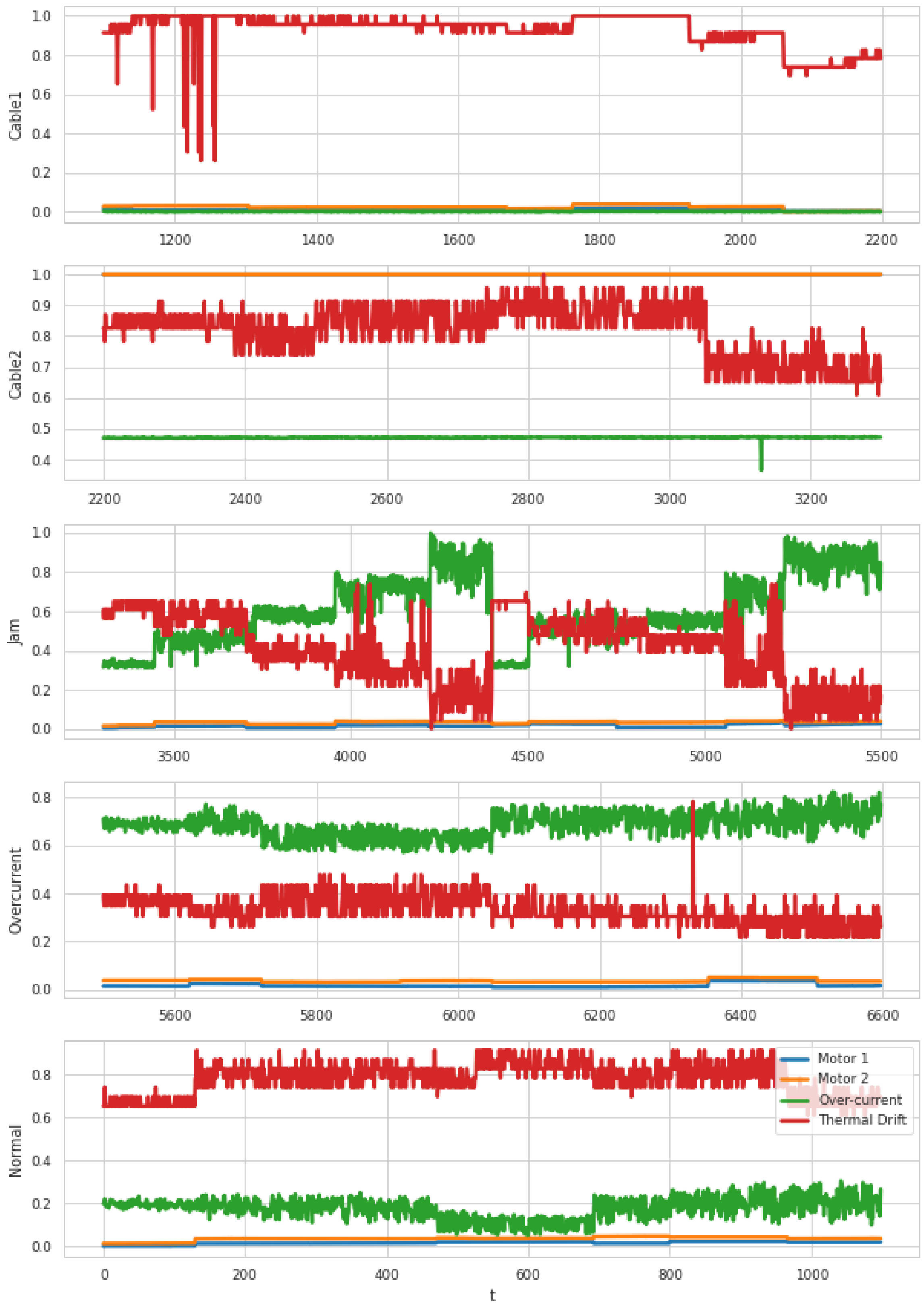


Figure 4.2: Sensor plots per class. Each row of plots show the collective sensor behaviour for each fault class. Each sensor is assigned a color. Output for Motor 1 is blue, output for Motor 2 is orange, Current is green, and Thermal Drift is red. The most descriptive sensor readings correspond to Current and Thermal Drift.

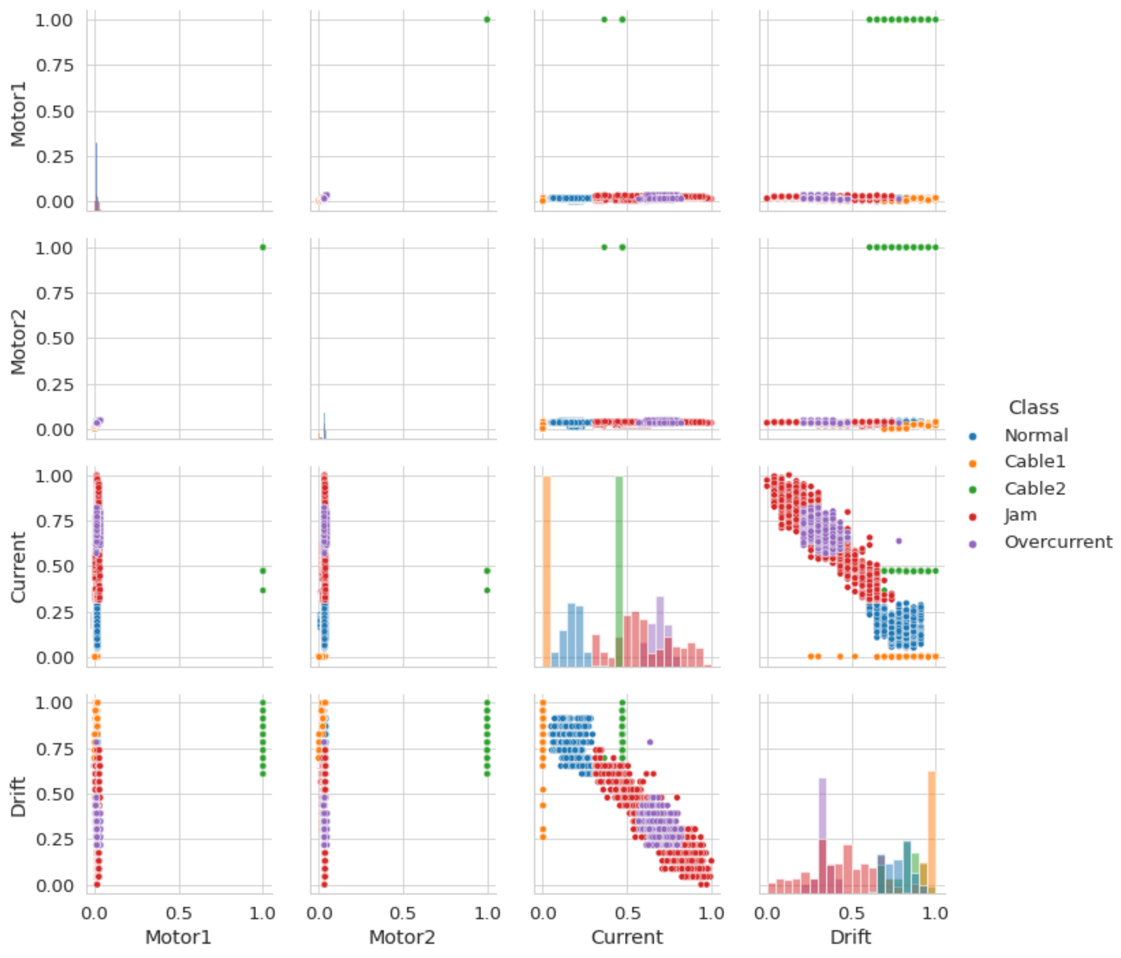
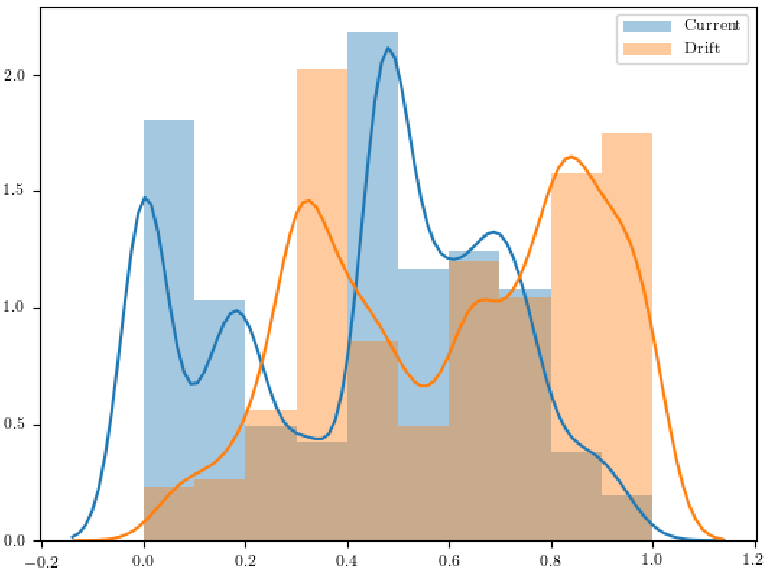


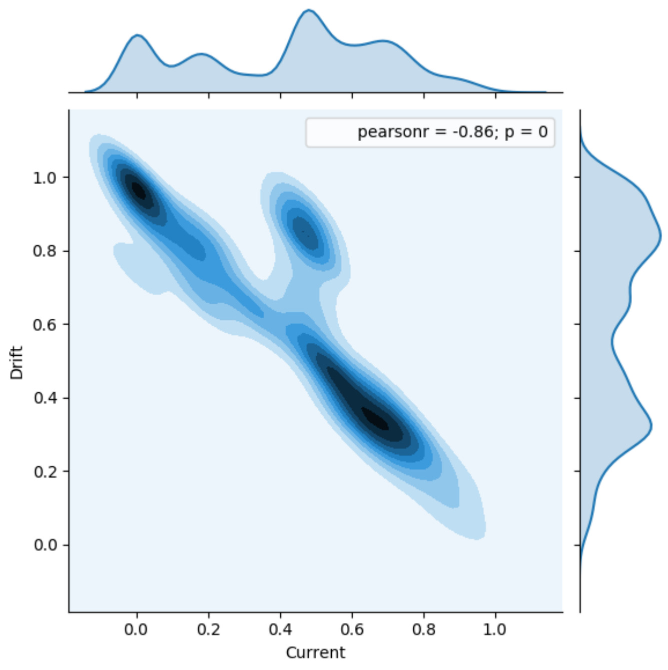
Figure 4.3: Pairwise relationships in the RECOVERY Testbed dataset. Densities and multi-modalities of the data pair are shown to understand their correlation. The diagonal plots in this figure are histograms for each sensor on the specific fault setting. The remaining plots are scatter density plots for a sensor pair, defined by the row and column, in a specific fault setting. The settings are expressed in colors as defined in the legend.

Figure 4.4 shows a closer look roughly estimating the densities and multi-modalities of the data pair. A joint density plot in figures 4.4 with a kernel density estimator better describes this correlation and shows in darker colors the densely populated areas with more clarity than the scatter plot in Figure 4.3. This also includes the Pearson correlation coefficient and the p-value.

The Pearson correlation coefficient varies between -1 and $+1$. The negative normal correlation value for this pair, $-.86$, states that as current, the value of thermal drift decreases. The p-value provides supplementary information to indicate the probability of uncorrelated data. In this case the value is very small, shown as zero. This suggests that either of these two readings could be on its own to avoid redundancy.



(a) KDE Estimation - Histograms



(b) KDE Estimation - Joint Distribution

Figure 4.4: (a) Kernel density estimations Current and Drift pair with $bins = 15$ and (b) their joint plot showing the correlation Pearson value of -0.86. The negative normal correlation value for this pair states that as one value increases, the other decreases. The p-value provides supplementary information to indicate the probability of uncorrelated data. In this case the value is very small, shown as zero. So, this suggests that either of these two sensor readings could be used on their own.

4.4 Candidate Architectures

The analysis done above helped in defining a strategy by selecting the most relevant features needed and identifying suitable methodologies. The problem of thruster system failure classification is performed using three machine learning methodologies.

1. A Gaussian Mixture Hidden Markov Model
2. A 1-D CNN
3. A Long Short-Term Memory Network

These approaches have been chosen to consider the chronological nature of the data. The Hidden Markov Model incorporates probabilistic strategies to infer internal states and transitions in sequential data. The CNN, mostly used in Image Processing problems, can also be utilized for time series. Similarly, The LSTM also models time series by maintaining persistence in the information as it learns, that way it can remember previous information about the short-term dependencies in the data. These methods are used to categorize the data from the RECOVERY dataset and the procedures are described in the following sections.

4.5 Gaussian Mixture Hidden Markov Model

The HMM is a variation of a Bayesian temporal model which is used for sequential analysis and representation. Although it is most commonly used for speech recognition applications, it is appropriate in this scenario where the motor behaviour classes are represented within a state space. It's formal definition according to Rabiner 1989 is a generative model that yields sequences in the form of a first-order Markov chain with the following elements:

- A sequence of observable variables, x, generated by
- a sequence hidden states of the model z.
- Sequence z is defined by an initial state vector, π, and
- a transition probability matrix, A.
- The observable output, x, which follows a probability distribution with parameters, θ, depending on the current state.

The analysis shown in previous section showed how the KDE yields multiple mixtures for current and thermal drift readings. Following this line of analysis, a GMM is defined as the probability density function for the HMM. A set of GMMs with different parameterizations were generated. The best fit or the optimal number of mixtures for the GMM-HMM is then chosen on the lowest Bayesian Information Criterion (BIC) score.

The BIC is an indicator used in model selection. The Akaike Information Criterion (AIC) is used also for the same purpose. Both methods are closely related to one another. As shown from the formulas below,

$$AIC = 2k - 2\ln(\hat{L}) \tag{4.1}$$

$$BIC = \ln(n)k - 2\ln(\hat{L}), \tag{4.2}$$

where $\hat{L}$ refers to the log-likelihood of the model on the training set, and k is the number of parameters in the model. In the case of the AIC, the n refers to the number of samples in the training set. Both are penalized-likelihood criteria for model estimators in a given set of data but focused on achieving different goals:

1. **Best prediction**. Finding the model that provides the most accurate prediction, assuming none of the models are correct.

2. **True model**. Assuming that one of the models is the *true* model.

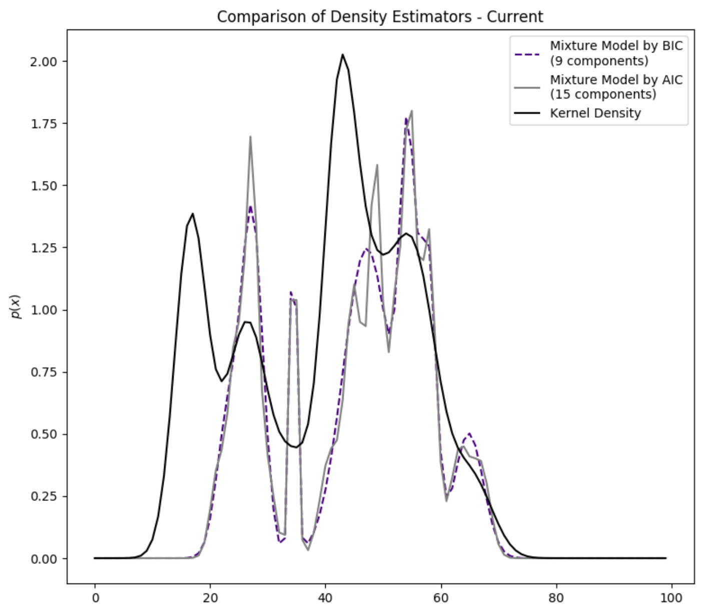
Commonly, the AIC and cross-validation focus in 1 and BIC focuses in 2 by penalizing more heavily. It has been shown that the AIC and cross-validation are asymptotically equivalent in Stone 1977; Fang 2011. Both approaches will be used for the selection of the model. The model will help in discovering the latent classes in the data. Because AIC is more inclined to choosing a larger model regardless of the number of samples, there the possibility that both Information Criteria (IC) could disagree.

AIC is better in situations when a false negative finding would be considered more misleading than a false positive, and BIC is better in situations where, in classification, a miss is more misleading than a false positive. On the other hand, BIC would be better in situations where a false positive is as, or more misleading than, a miss. Generally speaking, if BIC and AIC choose different size models it is recommended to select a range of models with those number of sizes.

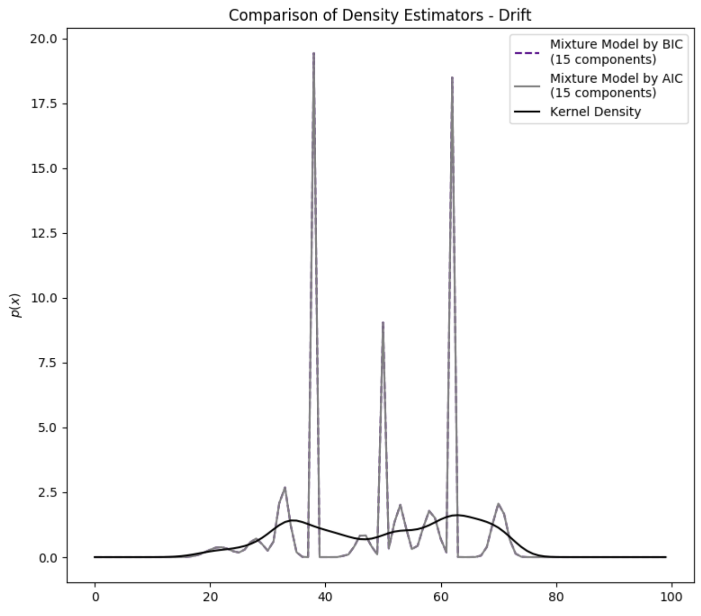
The comparison of density estimators shown in Figure 4.5 shows the optimal number of GMMs components for each sensor. The GMM solution for the current sensor exhibits similar densities to that of the KDE with 9 mixture components for both IC. In the case of the thermal drift, the optimal number of parameters is large, 15, for both IC and suggests over-fitting. Both configurations are tested on the RECOVERY dataset to help in establish the elements that will define the HMM.

Recalling the HMMs elements from the definition by Rabiner at the beginning of this section, the model is defined by π, A and θ. To find these, the HMM solves three fundamental problems:

1. Given just the observed data, estimate the model parameters.



(a) Density Estimators - Current



(b) Density Estimators - Drift

Figure 4.5: Comparison of a Kernel Density Estimator and a Gaussian Mixture Model for (a) Current and (b) Thermal Drift sensors. These plots illustrate how the estimators are better at estimating the distribution of the Current sensor versus Thermal Drift readings.

2. Given the model parameters and observed data, estimate the optimal sequence of hidden states.

3. Given the model parameters and observed data, calculate the model likelihood.

The first problem is solved by an iterative Expectation Maximization (EM) process known as the Baum-Welch algorithm. The second and third problems are calculated by the dynamic programming algorithms known as the Viterbi algorithm and the Forward-Backward algorithm, respectively.

4.5.1 Current Sensor

The KDE corresponding to the current sensor readings determined that the best number of mixture components for the HMM is 9. The number of hidden states suits the five thruster system behaviour categories: cable 1 and cable 2 interruptions, motor jam, over-current and normal.

The HMM is trained with 70% of the data and specified to have five hidden states with nine Gaussian components. While this is done in an unsupervised manner, the HMM is still manages to assign most of the categories correctly to its own hidden state. Figure 4.6 shows the test set, 30% of the RECOVERY dataset, assigned to five hidden states.

However, this HMM only achieved 79.35% accuracy. This is due to most of the motor-jam samples being misclassified as over-current.

4.5.2 Thermal Drift Sensor

The number of mixture components selected by both the AIC and BIC is 15. Figure 4.7 shows how the classes were decoded. In this case it is harder to identify visually compared to the current sensor HMM as the range of values distinctly corresponded to the true class.

4.6 1-D Convolutional Network

The CNN is used in combination with other types of NN layers in ML problems. Section 3.4.2 summarized how CNN filters scanned over a time series to create a feature map. It is common, but not required, that after each Convolutional layer is succeeded by a Pooling layer whose job is to down-sample the amount of features acquired to maintain the most relevant information.

Subsequently, a sets or a single layer, or FCN, will then take the output of the previous layers to flatten the information, i.e. formatting the data into a single vector for prediction.

The CNN architecture used for the following experiments consists of a single 1-D Convolutional layer, an Average Pooling layer and a FCN to flatten and classify using a softmax activation. The network was evaluated using both the current and thermal drift sensor data separately.

Figure 4.8 illustrates the training process. Using categorical cross entropy, the analysis of per epoch loss and cross validation accuracy is important as it gives an understanding

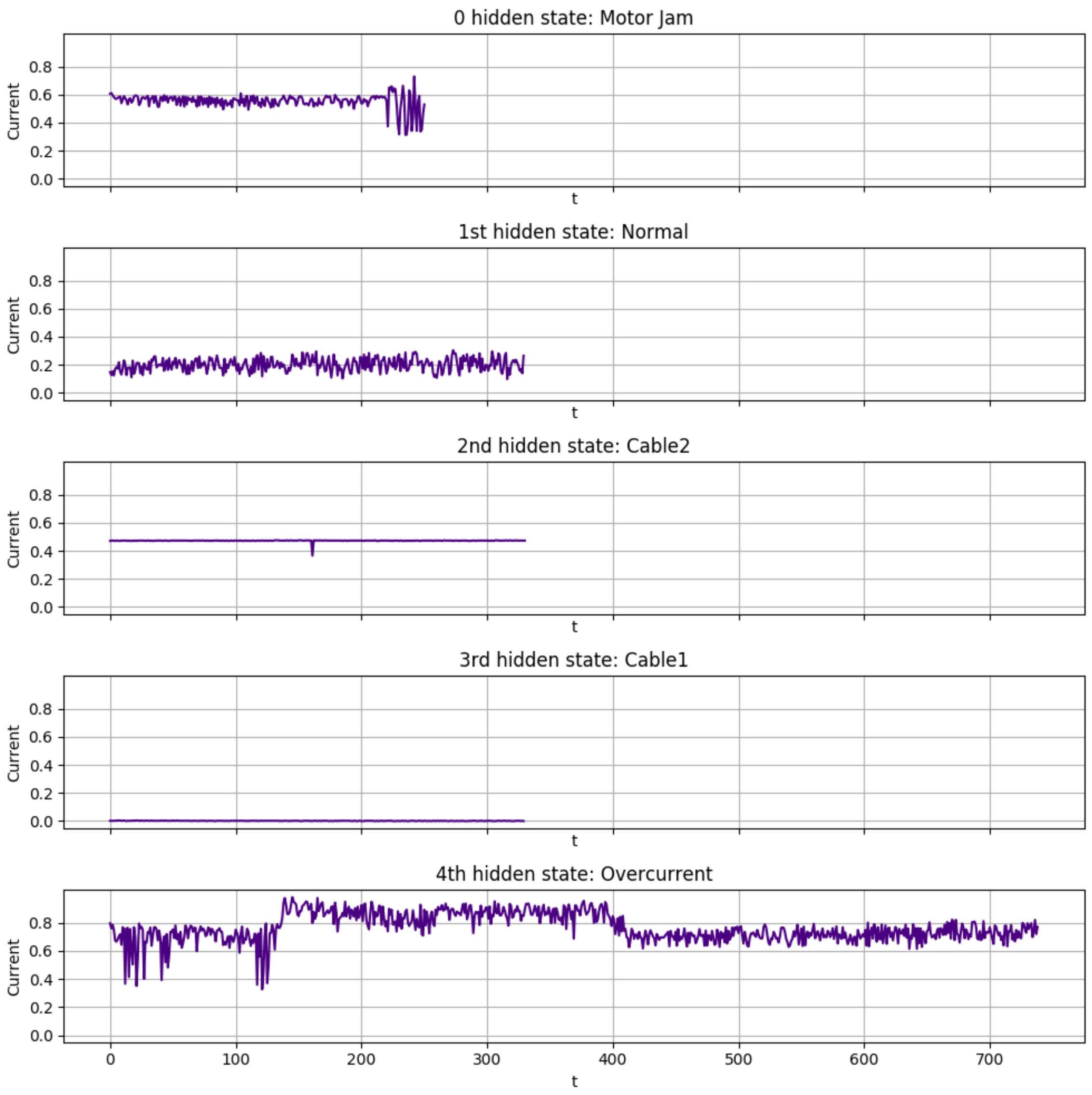


Figure 4.6: Hidden Markov Model trained by current sensor readings. The RECOVERY test set is assigned to hidden states corresponding to thruster system behaviour categories. Each row corresponds to a thruster behaviour and all correspond to the Current sensor as expressed in the *x* axis.

of the network convergence, precision and robustness. A summary of the scoring for these networks is listed in Table 4.2.

4.7 Long Short-Term Memory Network

The LSTM was evaluated in the same way as the CNN testing for two different sensor readings individually. Unlike a CNN, the LSTM does not require additional layers like Pooling, it does however requires at least one layer of a FCN for classification output using a softmax activation.

Keeping the same practices from the CNN, figure 4.9 shows loss and accuracy for cross validation per epoch. The next section discusses the performance of the three models on

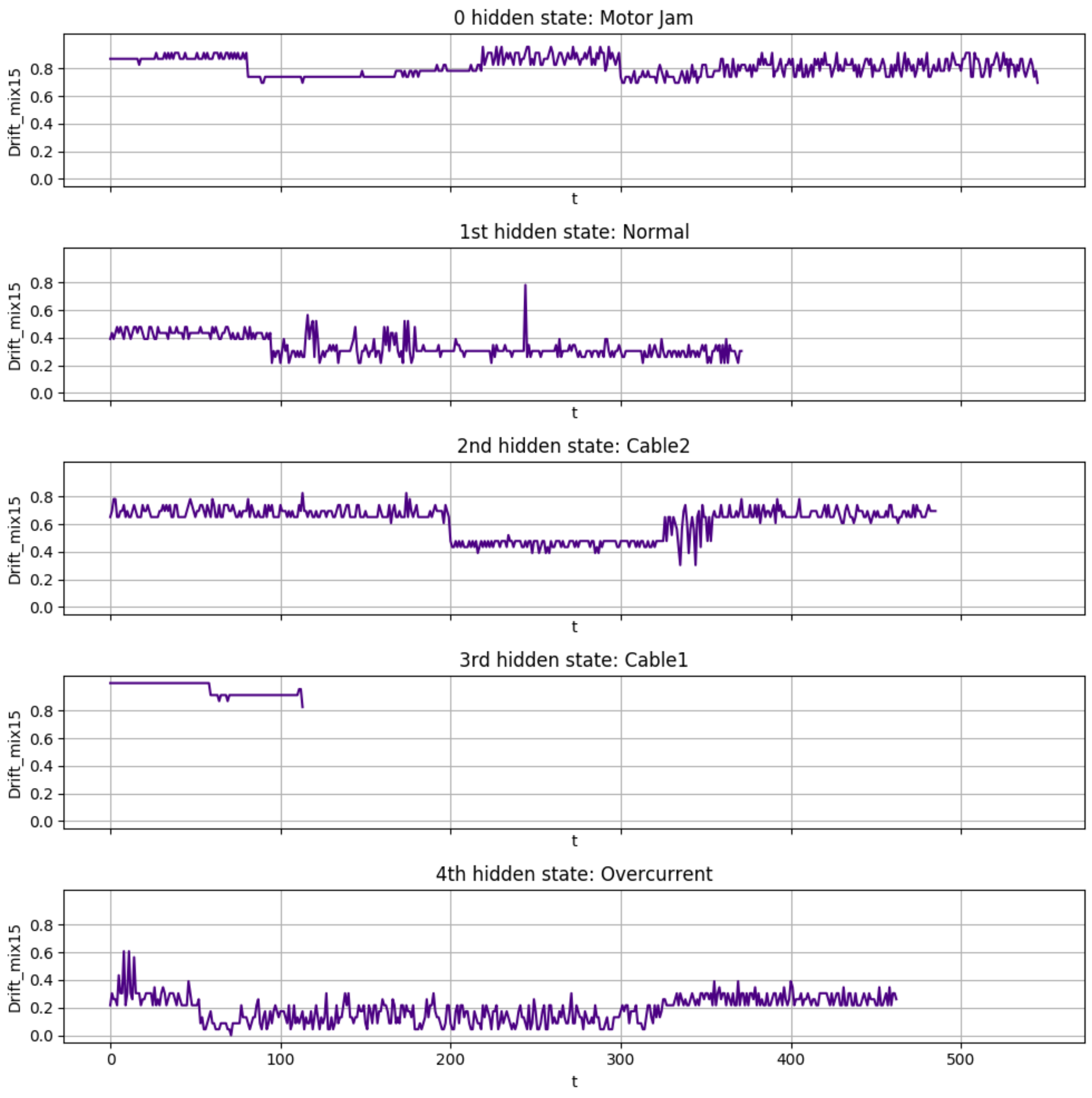


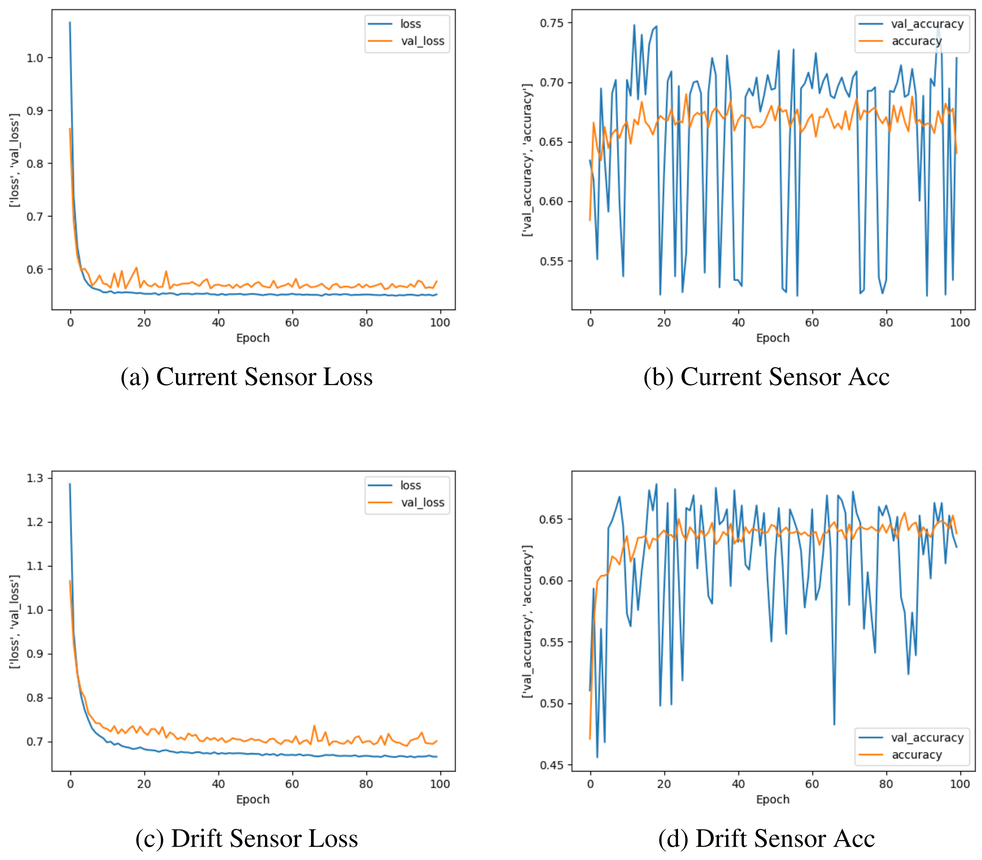
Figure 4.7: Hidden Markov Model trained by thermal drift sensor readings. The RECOVERY test set is assigned to hidden states corresponding to thruster system behaviour categories. Each row corresponds to a thruster behaviour and all correspond to the thermal drift sensor as expressed in the x axis.

the RECOVERY data.

4.8 Results

This section reviews the performance of the ML methods used for classifying thruster system behaviours. In overview, the GMM-HMM performed poorly compared to the NN-based results, however it was still able to model with some error the different categories of the system.

While it is recommended that hyper-parametrization tuning is done when training NN architecture, these were tuned with a series empirical tests. This was primarily due to the initial statistical analysis which led to the discovery of important single sensor features that



(a) Current Sensor Loss (b) Current Sensor Acc

(c) Drift Sensor Loss (d) Drift Sensor Acc

Figure 4.8: CNN Loss and Accuracy per epoch.

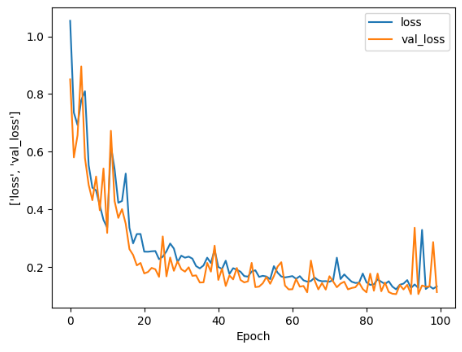
described the behaviours.

GMM-HMM Having different number of mixtures by the information criteria prompts to evaluate a range of mixtures with various number of sizes. Table 4.2 summarizes evaluation scores for these models. In addition, Figure 4.10 illustrates confusion matrix plots for both HMMs by current and thermal drift sensor readings.

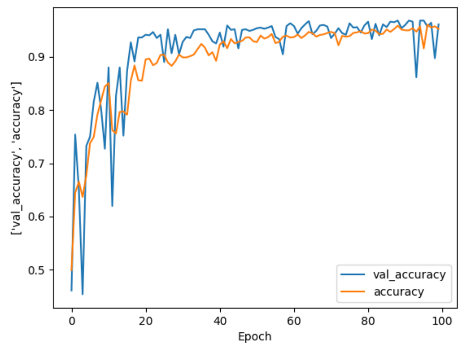
These results suggest that the GMM-HMM is not very robust at decoding the data as a sequence. While most categories show trend in values as specified in Table 4.1, this is not the case for motor jam, and therefore it is often mistaken for over-current.

It is concluded from these results that choosing the current sensor is more representative of thruster behaviour for categorizing the different known behaviours of a thruster system function than thermal drift. The misclassification of motor jam is could be caused by the overlapping distributions of the latter with over-current. Though it could be fundamentally justified that it is an over-current fault due to a motor jam. Therefore it could potentially improve the results if these these two categories were merged when modeling these behaviours using a GMM-HMM.

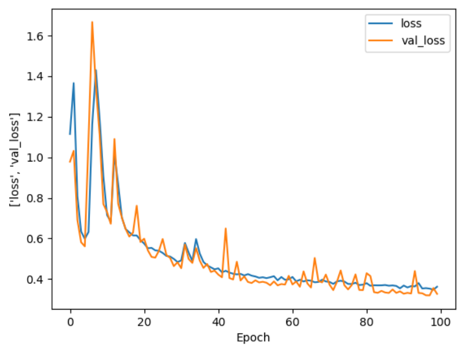
While using the current sensor in this architecture proved to have the most representative features needed for categorizing these faults, the thermal drift sensor could be useful for



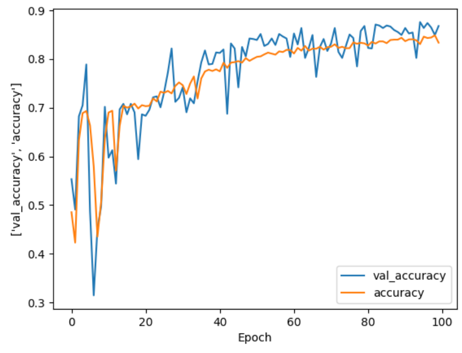
(a) Current Sensor Loss



(b) Current Sensor Acc



(c) Drift Sensor Loss



(d) Drift Sensor Acc

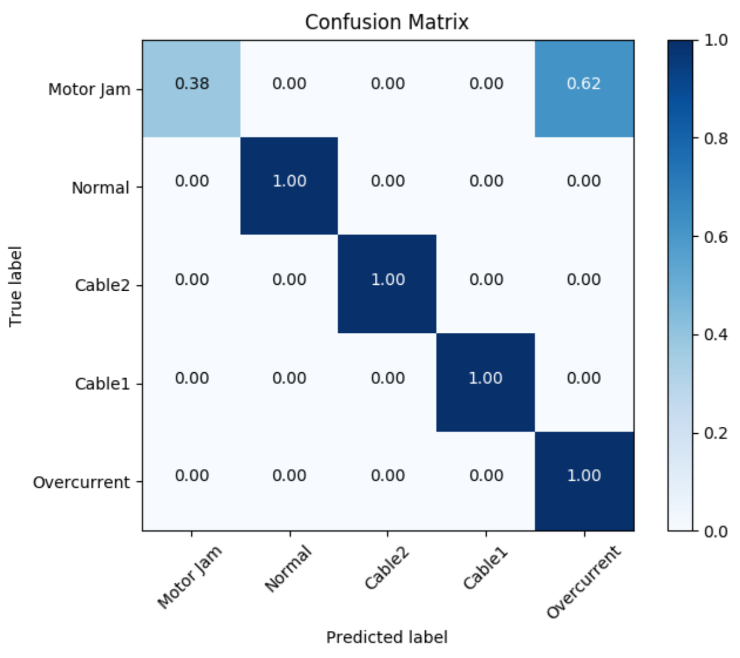
Figure 4.9: LSTM Loss and Accuracy per epoch.

binary classification, given that the temperature representation in volts is more distinctive between these two classes. However, this is not explored in this work as the main motivation is to determine the cause or the faulty behaviour in the system.

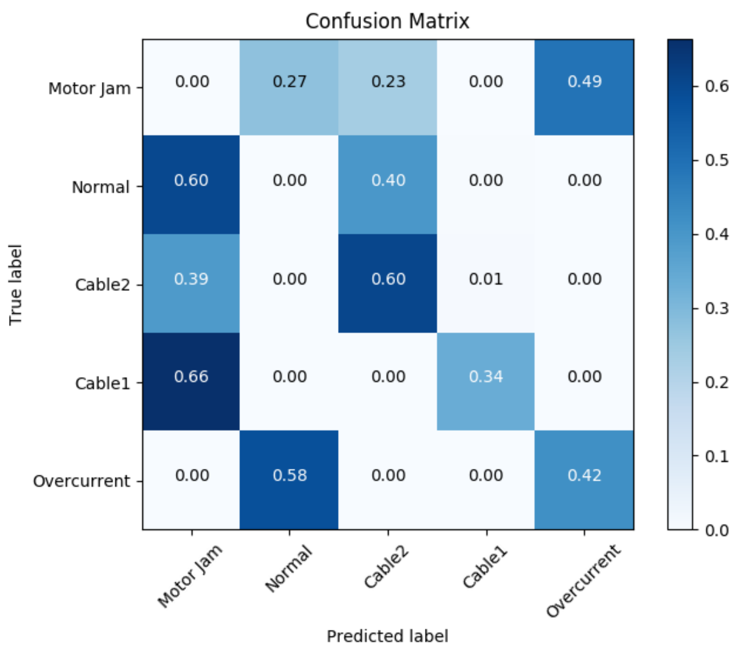
Sensor	**No. Mix**	**Accuracy**	**Precision**	**Recall**
Current	9	79.35%	88.93%	87.60%
Thermal Drift	15	22.66%	33.66%	27.17%
Thermal Drift	12	19.88%	33.16%	23.84%
Thermal Drift	9	12.01%	30.76%	14.39%
Thermal Drift	5	20.29%	33.61%	24.33%
Thermal Drift	3	21.17%	33.53%	26.08%

Table 4.2: GMM-HMM Results

CNN and LSTM As stated above, these NNs were defined without hyper-parametrization tuning, i.e. the settings of the networks were defined based on the length of the sequence: 100 time steps, and empirical hyper-parameters. Training with a total of 3,902 samples and validating on 976, a 80%-20% percent data split, it is



(a) Current



(b) Thermal Drift

Figure 4.10: Confusion Matrix for Current and Thermal Drift HMMs.

guaranteed that the number of parameters does not surpass the number of training samples, which is a general recommendation to avoid model over-fitting.

As such, the kernel size for the CNN was set to 16 to accommodate at least 10% of the sequence in the convolution, as well as the number of filters. The LSTM was also defined to have 16 cells. In total, the CNN has a total number of parameters of 357 and the LSTM 1,237.

Table 4.3 shows the test scores for each sensor experiment. Each were run for 100

Network	Sensor	Accuracy	Precision	Recall
CNN	Current	74.59%	66.19%	74.76%
CNN	Thermal Drift	66.14%	54.86%	60.11%
LSTM	Current	95.49%	95.62%	96.33%
LSTM	Thermal Drift	85.16%	84.29%	85.14%

Table 4.3: CNN and LSTM Results

epochs as shown in the Loss and Accuracy plots. The convergence rate for both networks also shows that, given these hyper-parameters, the number of epochs could be reduced by half, as it appears to reach an acceptable validation accuracy around the 40th epoch.

The CNN shows average performance, similar to the HMM, however from the number of parameters, gives margin for a larger and more robust network. The LSTM surpassed both networks with impressive scores above 95%.

Despite the simple nature of the problem, the three ML models clearly showed their capabilities. Undoubtedly, the CNN can be improved by standard hyper-parametrization techniques. Nevertheless, the LSTM illustrated that for this very simple task, it outperformed a well-established robust ML learning methodology: the HMM.

4.9 Summary

This chapter presented the implementation of three fundamental ML methods, the GMM-HMM, the CNN and LSTM, in the context of a classification problem for the RECOVERY dataset. This problem required these methods to robustly categorize the different types of Thruster System behaviour. The main motivation is to enable an AUV with enhanced autonomous capabilities, such as being able to detect hardware induced anomalies.

The three methodologies are chosen based on their capability to handle time series and sequential data. The assessment of their performance was based on the observation of required initial data analysis where it was pertinent, convergence, precision and robustness as well as the computational complexity of the NNs.

The contributions of this work update the state-of-the-art in the topic of sensor fault characterization for thruster malfunction isolation. It was observed that due to the simple nature and attributes of the data, that the NNs surpassed the HMM with very little hyper-parameter tuning. The outcome of these experiments demonstrate that a shallow 1-layer LSTM network with 16 nodes, a total of $1,237$ parameters, can outperform a GMM-HMM classifier. While recent works implement NNs for other AUVs applications, the results presented in this work outperform those described in the literature review for sensor fault characterization for thruster malfunction literature. Namely, the works by Rae and Dunn 1994; Qin and Gu 2009 by achieving robust results with documented metrics.

In addition, this work outperforms the work by Ranganathan et al. 2001 which proposed a 2-layer ANN with layer sizes $L = \{l_1 : 16, l_2 : 32\}$ and a rule-based inference system that yielded an average accuracy of 93%. This work proposes a single layer LSTM-based NN of size $L = \{l_1 : 16\}$ achieving scores of 95.49% accuracy, 95.62% precision and 96.33% recall.

Part of the work described in this Chapter was presented and included in the proceedings of the 2016 OCEANS MTS/IEEE Conference, De Lucas Alvarez and Lane 2016.

Chapter 5

Task-Oriented Classification

"As I have evolved, so has my understanding of the Three Laws."

-V.I.K.I, *I Robot*

The identification of events requires close examination of the robots processes as they take place. Tasks, when executed by autonomous robots, are progressive conditions that are composed of multiple consecutive time instances from a single or various sensors rather from a sole time step. In contrast to hardware-oriented, task-oriented classification is based on understanding the current state of the robot while its interacting with the environment or executing its tasks rather than observing its internal functionalities. Nevertheless, it is the obligation of the problem formulator to define the nature of a task.

Tasks can be described as sets of behaviours that vary in execution and procedure. This requires that the method of modeling such tasks keeps the chronological dependencies to maintain the meaning of the process. The previous chapter included an HMM for component fault classification. While the problem may seem similar in essence to the one postulated in this Chapter, there are important nuances that differentiate task-oriented from the latter. As the name states, HMMs maintain Markovian assumptions in which current state is dependent on past states, e.g. first order Markov assumption states the current state is dependent only on the past time step. For one thing, this could provide us with a model with more memory, but it would result in a more complex model and could potentially become intractable. In addition, the state transitions are predefined in HMMs, i.e. the design of the model becomes subject to human error if by some situation the expert overlooks some flow of information and therefore the model becomes susceptible to misclassification in highly complex problems.

LSTMs are better suited for task-oriented classification because they are not limited to the above criteria. These networks are from the family of RNNs that have resolved the concern for vanishing/exploding gradients (Bengio, Frasconi, et al. 1993; Bengio, Frasconi, et al. 1994; Hochreiter 1991) which are responsible for maintaining long short-term dependencies in time series data. Furthermore, LSTMs in theory can learn the patterns in the input data that represent state transitions and thus it is unnecessary for these to be defined beforehand. The applicability is delineated in the following section.

5.1 Application: Navigational Trajectory Classification

Considering that mission analysis in autonomous robots requires methods that are capable of handling sequential information, this Chapter presents experiments where trajectory classification plays a crucial role in a real world application. For autonomous vehicles, a fundamental task is navigation which is vital for performing a survey or exploration in water or ground. Working with autonomous vehicles requires techniques dedicated to efficiently reporting health and status of a task or mission. Accurate provision of this information is a vital component in field operations.

The following experiments use navigational data to interpret trajectory patterns and classify them. The methods are focused on LSTMs to learn the most commonly used survey trajectory patterns and contrast their performance against standard ML methods.

Context In marine robotics, AUVs are frequently required to survey a large underwater perimeter with objectives. The most common tasks during these missions is to search for objects of potential interest or construct a map of the ocean floor using its imaging payload. Two characteristic navigational trajectory patterns for these tasks are the *Spiral* and *Lawnmower* respectively. Mission monitors can easily determine the current status of the mission as the trajectories can be viewed as it unfolds. However the AUV simply executes the way points it receives with no knowledge of its current task. Embedding this ML based AI to the vehicle would give valuable information in mission failure detection where access and resources are limited. It is essential for enabling successful missions and for maintaining the health of an AUV that is most of the time, if not all, submerged and out of sight.

Solving this problem entails that the chronological dependencies of each advancing position are kept in order for each position sequence to be categorized. The observed trajectory or task is defined as the category since the trajectory pattern is specific to the function the AUV is executing. Keeping the central goals present, this chapter addresses these research questions:

1. How deep does the network need to be to robustly categorize each trajectory?
2. Given that the models can take in variable lengths of input, how short can the sequences be and keep a robust model?

While the problem can be approached in an unsupervised way, the available knowledge from a specific mission or routine can enable sample engineering and labeling. The REGIME dataset helps to exemplify these types of navigational trajectories and demonstrates the benefits of trajectory classification for self-monitoring by navigational behaviours during mission execution. The following section is dedicated to describe in detail this dataset.

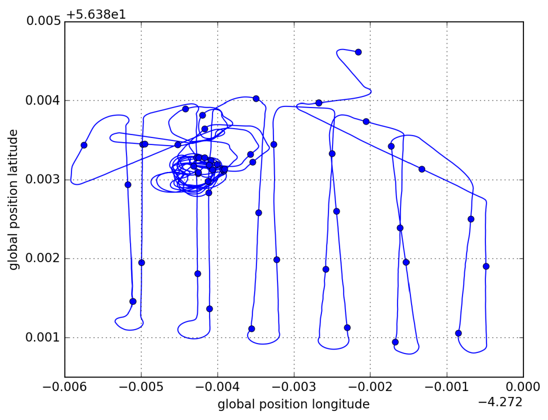
5.2 REGIME Dataset

The dataset is a collection of navigational data from two AUVs, IVER and REMUS, executing a series of survey missions in Loch Earn, Scotland. The data was obtained using the Neptune operating system from software company SeeByte and totals 23 hours of operation. Although the mission logs contain varied recordings from different systems, only the navigational data is being used. Each log comprises a full survey scan of a predefined area of the Loch which contains various sensor readings. From these readings six numerical features have been selected to construct a sequence $s = \{Lat, Lon, Depth, Att_x, Att_y, Att_z\}$. These are position measurements in latitude, longitude and depth, and vehicle attitude in three axis. There is no missing data, i.e. there are no dropouts or unavailability of the the sensors.

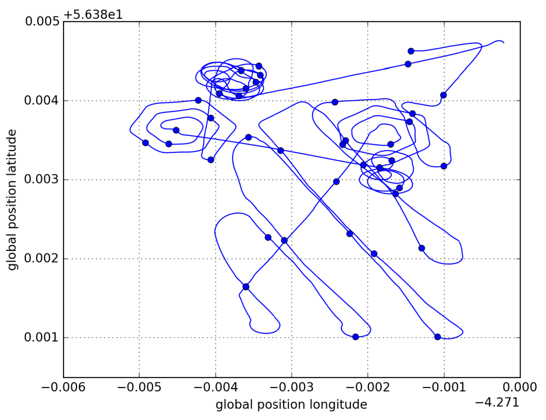
The dataset was used to extract basic or commonly known path trajectories for surveying large areas of seabed for task monitoring and prediction. The dataset contains two classes of path trajectories:

1. **Spiral**. A path that is executed by an AUV when inspecting an object of potential interest. The AUV fixes its heading to the location of the object and navigates sideways, creating a spiral pattern.

2. **Lawnmower**. A path that is executed by the AUV when scanning over a wide are in the attempt to cover its totality.



(a) Vertical *Lawnmower*



(b) Diagonal *Lawnmower*

Figure 5.1: Two survey missions are illustrated here where the AUV performs two different survey missions. The AUV mainly executes two trajectories: Lawnmower and Spiral.

These classes are exemplified in Figure 5.1 where two different missions illustrate these two classes. An important attribute to note is that the *Lawnmower* pattern can be executed at different orientations. In Fig 5.1 two cases are shown where (a) is executed in a vertical fashion and (b) is done diagonally. Some spiral trajectories are shown as well.

The learning problems associated with dataset pertain to the task rather than the vehicle. There is also no mission failure information available. More details into how the learning problems were formulated and number of samples per class will come in the following sections.

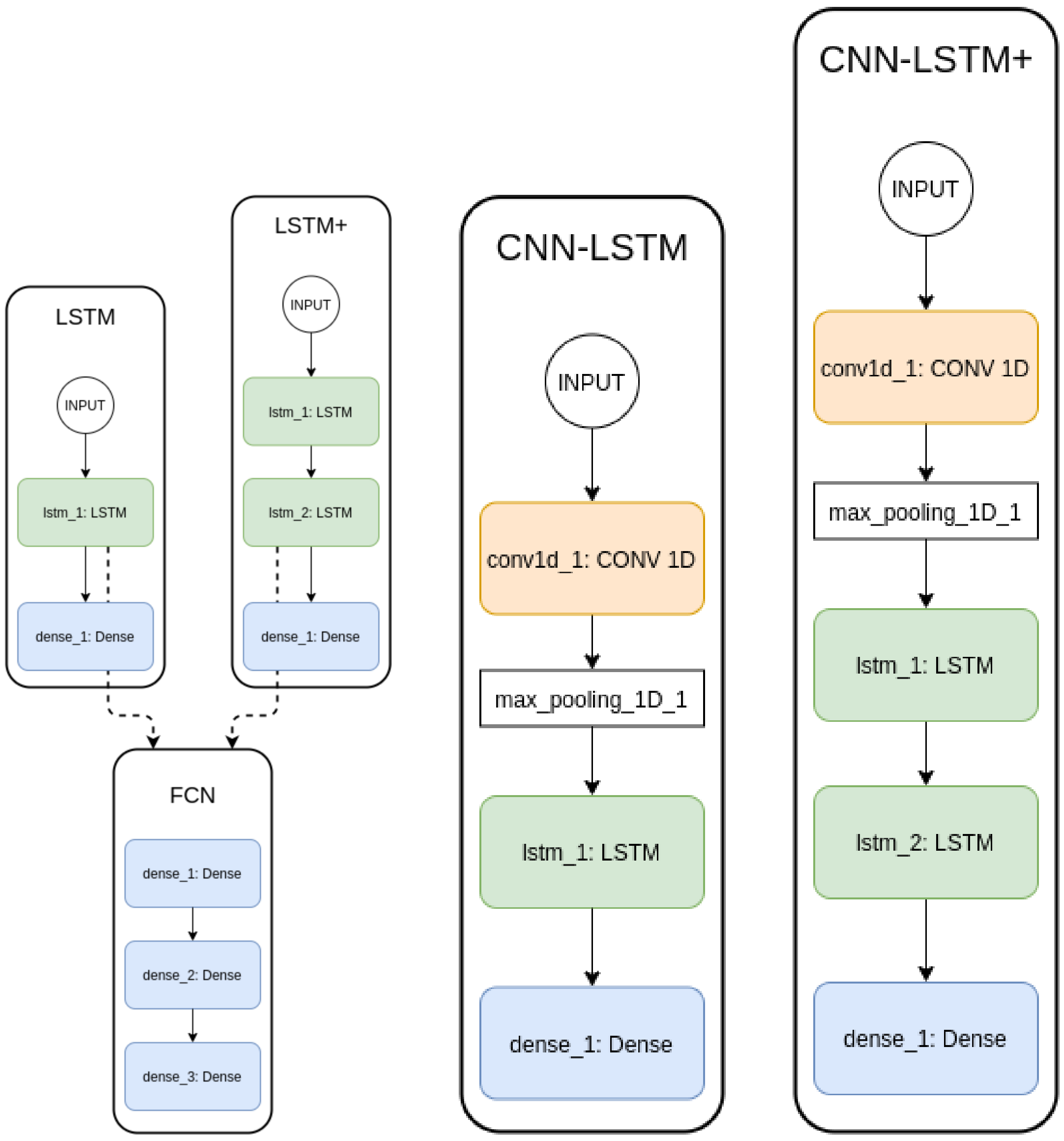
5.3 Candidate Architectures

The trajectory classification was evaluated on six different LSTM architectures. These were designed and parametrized to observe how the robustness adjusts with different criteria. Figure 5.2 shows the construction of these illustrating the concatenation of some smaller architectures. This yields in total six network architectures that are used repeatedly to perform experiments with variations in class weight. The list below provides more details into the construction and design of each architecture.

1. **LSTM.** A single input LSTM layer and one single Dense layer with a sigmoid output.
2. **LSTM+.** A 2-layer LSTM network with a sigmoid output.
3. **FCN.** A Fully Connected Network (FCN) of 3 ReLU-activated layers with a sigmoid output. This network is also used to stack the LSTM and LSTM+ generating LSTM-FCN and LSTM+-FCN respectively.
4. **CNN-LSTM.** A 1-layer CNN connected to the described LSTM network.
5. **CNN-LSTM+.** A 1-layer CNN connected to the described LSTM+ network.

For Figure 5.2a shows the LSTM, LSTM+ as blocks that can be connected to the FCN to generate a new network, as illustrated but the dotted lines. The sigmoid output are hence removed from the LSTM and LSTM+ blocks and take the inherit the sigmoid output of the FCN. Recalling Chapter 3, sigmoid activations apply when working with binary classifiers.

The combinations of certain networks are motivated to evaluate the differences between networks that have one or more Dense layers, a dropout regularizer, and class balancing. Figures 5.2b and 5.2c show the stacking of CNN and LSTM networks. The CNNs are comprised of 2 layers 1-Dimensional Convolutional operations. The citation to sequence classification architectures mentioned in Section 3.1 (one-to-one, one-to-many, etc.) only concern to RNNs. CNNs learn spatial information, ergo this characteristic does not exist. However, this is motivated by the CNNs ability to effectively represent spatial information. In the context of time-series, the 1-Dimensional convolution allows for the parameters to



(a) LSTM, LSTM+ & FCN (b) CNN-LSTM (c) CNN-LSTM+

Figure 5.2: LSTM architectures. (a) LSTM, LSTM+ and FCN architectures shown as as stackable blocks. (b) A 1-D CNN stacked with a LSTM block. (c) A 1-D CNN stacked with a LSTM+ block.

be shared across time. This sharing of parameters outputs a time line to denote at what moments certain features emerge at the input. These networks have a one input CNN and one hidden CNN layer. The stacking allows for a lower dimensional feature map be used as input for the LSTM network variants which all have Dense sigmoid output for binary class prediction.

The implementation of regularization techniques repeats the number of evaluations of our architectures. For instance in the case of *Class Balancing*, an initial experiment is performed to demonstrate the effect of training the most simple model, the LSTM, when the class weights are not adjusted. Differences in performance is also assessed with including *Dropout* layers to some of our networks. Both *Class Balancing* and *Dropout* are explained in more detail in the following sections.

5.4 Class Balancing

It is often the case in real world problems were there are very few examples of a particular class. Several methods exist to mitigate classification errors caused by class imbalance. While some techniques focus on misclassification problems, some are also aimed at dealing with identifying minority classes. As with any ML methodology, performance metrics are important. However, in the case class imbalance it is important to not exclusively rely on accuracy and observe other metrics such as the confusion matrix, precision and recall. Other model specifications can be derived from these such as Area Under the Receiver Operating Characteristics (AUC-ROC), sensitivity, specificity and F1 score.

The size of the sequence defines the total number of samples in the dataset. Having shorter trajectory segments yields more examples. However, *lawnmower* trajectories are usually executed in larger survey areas than *spiral*. Recall the chosen segment length of 50 time steps. Figure 5.3 shows the class percentage.

This motivates the use of metrics other than accuracy when tuning the parameters for training. In addition, class balancing through setting class weights to each sample. This allows for performing unbiased training without losing training data. Naturally, this can be implemented when working with LSTMs. In this manner, the training process is balanced by setting higher importances to underrepresented classes.

The heuristic used for setting balanced class weights is based on the work presented in King et al. 2001 and is given by:

$$W_c = S_t / C_t * S_c \tag{5.1}$$

Where S_a is the total number of samples in the dataset, C_t is the total number of classes and S_c is the number of samples for class c. A second heuristic procedure for assigning proportional class weights is simply to weight up the class with fewer samples proportionally to its underrepresentation:

$$W_{c_s} = S_{c_g} / S_{c_s} \tag{5.2}$$

Where W_{c_s} is the weight for the smaller class, S_{c_g} is the total number of samples for the greater class and similarly for the smaller class, S_{c_s}. Table 5.1 summarizes the total number of samples for this sequence length. It is expected that the model without any weight specification will have a harder time classifying *spiral* trajectories.

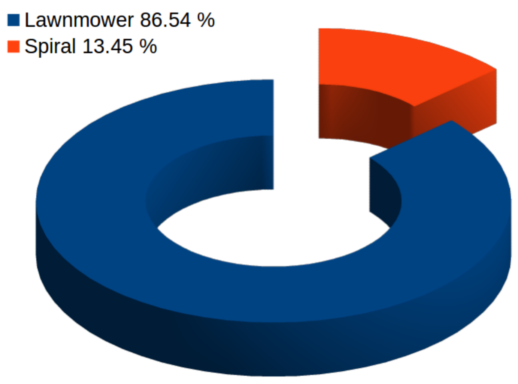


Figure 5.3: Sample percentage per trajectory type in the dataset.

Class	All samples	Train samples	Balanced weight	Proportional weight
Lawnmower	1114	746	0.58	1.0
Spiral	173	116	3.72	6.43
Total	1287	862		

Table 5.1: Number of samples per class in the REGIME dataset for sequences with length of 50 time steps. This information is used to calculated two sets of weights. Balanced weight is based on the heuristic by King et al. 2001. Proportional weight is defined by setting proportional importance values

5.5 Effect of Sequence Length & Dropout

The effects of sequence length and dropout are evaluated initially to asses if these parameters should be incorporated into the main experiments.

5.5.1 Sequence Length

The first criterion that was assessed was the length of the sequence. The trajectory of the vehicle is given as a set of navigation messages $x^0, \ldots, x^T$. When using LSTMs, variable lengths of sequences can be used, i.e the time step index ranges from 1 to T. Experiments were conducted to evaluate the effect that segmentation produces because, indistinctly of the category, the trajectories in the REGIME dataset vary in lengths.

First, a sequence is defined as the time series array representing a a trajectory segment of the original trajectory of size t time steps for any given type. The length of the sequence is constructed by sub sampling the full mission sample with different non-overlapping window widths: $100t$, $50t$, $20t$, $15t$, $10t$. These were evaluated in three different LSTM architectures. This choice of window size is selected to take into account small long-time variation effects in the sequences. This sequence formatting accommodates the Multiple-Input Single-Output or Many-To-One format of classification discussed in Section 3.1, as the predictor naturally needs multiple steps or inputs to classify the trajectory

	Search Domain
Fixed Settings	
LSTM Layer size	100
Class Balancing	Proportional
Searched Settings	
optimizers	[rmsprop, adam]
epochs	[50, 100, 150, 200]
batches	[28, 32, 64]
init	[glorot uniform, normal, uniform]
dropout w	[0.2, 0.25, 0.3, 0.5]
filters	[28, 32, 64]
filter lengths	[3, 5]

Table 5.2: Hyper-parameters for LSTM networks with Dropout

Model	100t	50t	20t	15t	10t
LSTM	95.92%	96.38%	94.57%	95.85%	96.49%
LSTM-FCN	87.76%	94.83%	93.52%	91.62%	92.92%
CNN-LSTM	91.33%	97.16%	95.51%	95.54%	96.28%
Acc Means	91.67%	**96.12**%	94.53%	94.33%	95.23%

Table 5.3: Selecting time step length based on Test set accuracy rates.

segment.

The Figures 5.4 show examples of how the segmentations with the different sequence lengths look like, taking as example two missions that show both types of navigation patterns. More trajectory examples can be found in Annex A.1.

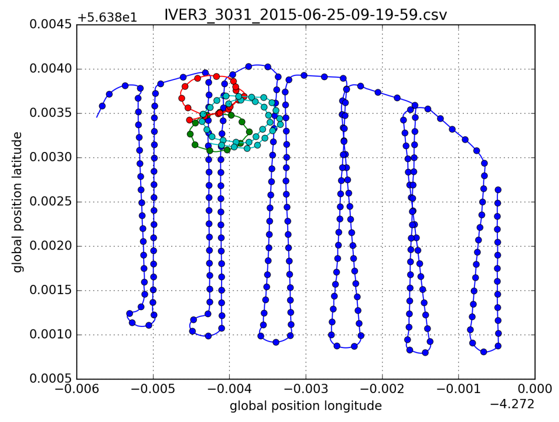
Three candidates are evaluated to observe the effect of the sequence length: *LSTM*, *LSTM-FCN*, and a *CNN-LSTM*. Table 5.2 shows the fixed and searched settings. Some parameters were not included in the search array and are set to fixed values like input layer size and class balancing.

The sequence length assessment was evaluated by training the Grid Search output for the specified window sizes. Table 5.3 shows the classifiers accuracy. The sequence length of 50 time steps recorded the highest accuracy.

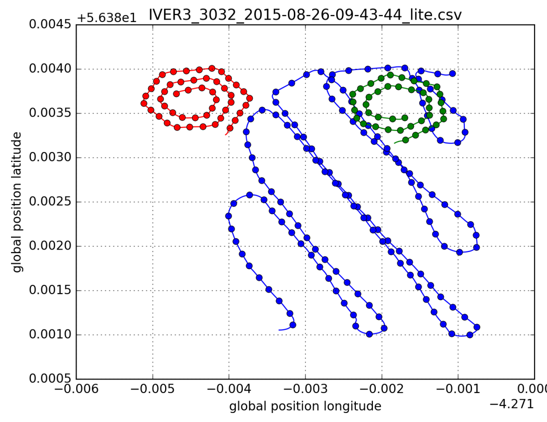
5.5.2 Dropout

Recommendations have been established in Srivastava et al. 2014 for setting the dropout unit percentage, where the classical range for Dropout units in Dense layers is to have a value that is closer to 1.0 in the input layer as to retain the most values for input and 0.5 for the hidden layers. In this case, the dropout is applied to the LSTM layer. The same range of parameters from Table 5.2 are applied in these experiments.

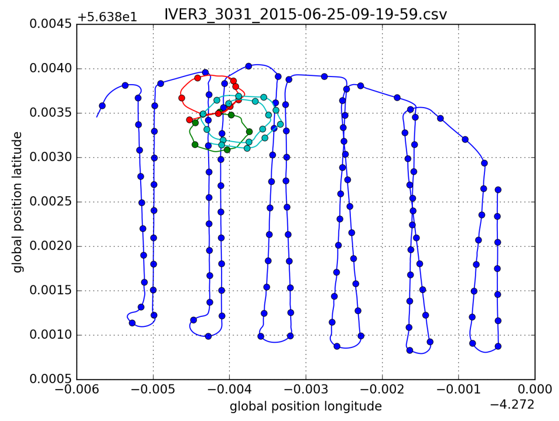
The performance of the LSTM networks are compared with four baseline classifiers.



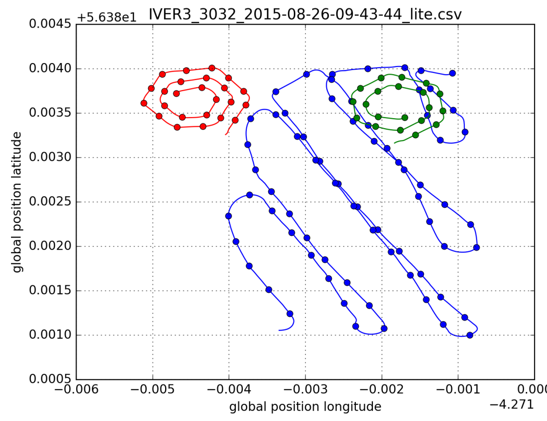
(a) 10*t*



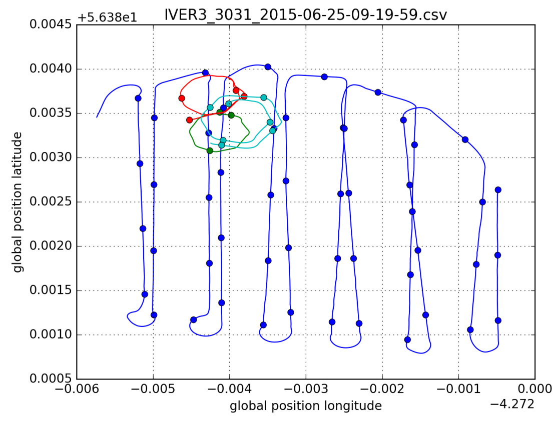
(b) 10*t*



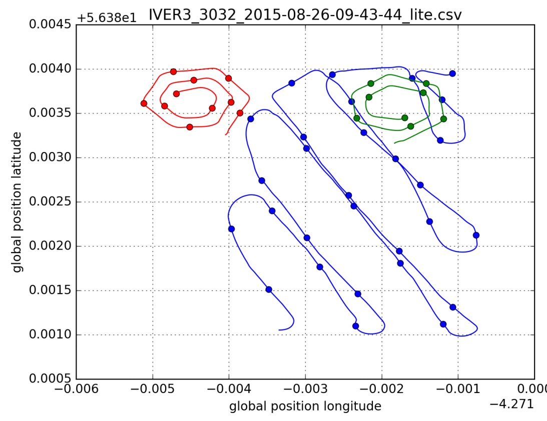
(c) 20*t*



(d) 20*t*



(e) 50*t*



(f) 50*t*

Figure 5.4: Example of two survey missions with different sequence lengths depicting both types of trajectory patterns: *Lawnmower* and *Spiral*. The survey patterns have been isolated from the rest of the mission and are highlighted in different colors.

The selection of these was based on the literature referenced in Chapter 2. Such are a Gaussian HMMs and Linear SVMs based on their usage in Nascimento et al. 2010;

Mlích and Chmelar 2008 and Biljecki 2012. Random Forests and Decision Trees were also included to take into account the clustering and data structure classification methods discussed in Lee, Han, Li, et al. 2008; Panagiotakis et al. 2009.

Small adjustments needed to be made to the data for the evaluation with the baseline classifiers. The input format is required to be in 2D and the REGIME dataset was formatted as 3D for the LSTMs. The data was simple unrolled the into 1D arrays to keep the sequential nature of the samples. This maintained the number of examples used but increased the feature space. The Gaussian HMM did not require this reformatting as it is capable of handling sequences. In the same manner, a 10-fold cross validation training was done for each of the baseline models.

The same class weight proportion was used for the baselines except the Gaussian HMM as individual classifiers were trained for each class. Table 5.5 shows the metrics obtained from test set evaluation. The F1 score and test set accuracy of the best LSTM is closely comparable with the Random Forest model.

It was observed that by introducing a Dropout layer of to the LSTMs showed a reduction of approximately 32% against the other combinations. Adding the Convolutional layers, however, roughly improves the accuracy. Reduced accuracy in the LSTM with dropout was unexpected as it is meant to prevent over-fitting, but later confirmed previous research findings that dropout is usually not recommended for time series as it suggests that relevant data is lost during learning (Lipton et al. 2016).

Model	Initializer	Optimizer	Epochs	Dropout	Batch	Filters	Kernel
LSTM	glorot Uniform	rmsprop	150	-	64	-	-
LSTM w/Dropout	normal	rmsprop	150	0.5	32	-	-
CNN-LSMT	uniform	rmsprop	150	-	28	32	3

Table 5.4: Best Hyper-parameter values from Grid Search Cross-validation.

Networks	Test Acc.	Precision	Recall	F1
LSTM	0.974	0.970	0.970	**0.970**
LSTM w/Drop	0.591	0.900	0.590	0.650
CNN-LSTM	0.925	0.920	0.920	0.920
Baselines	Test Acc.	Precision	Recall	F1
G-HMM	0.909	0.900	0.910	0.900
L-SVM	0.870	0.930	0.870	0.890
DT	0.945	0.960	0.950	0.950
RF	0.987	0.990	0.990	**0.990**

Table 5.5: Results of Accuracy, Precision, Recall, and F1 score on the test set for the LSTMs and baseline methods. Baselines are: Gaussian HMM, Linear SVM, Decision Tree, and Random Forest.

5.6 Effect of Balancing and Hyper-parameters

For these experiments Randomized Search was used for more efficient parametrization search. As with all parameter search methods, rather than speculating what values might work best for the networks, the parameter search makes it possible to try different combinations of values within a specified range for different types of parameters, thus generating a set of candidate networks with different characteristics.

The fixed settings in these experiments differ from the last because the effect of the balancing method needs to be observed. Additionally, the initialization was also removed from the parameter search and fixed to a *Glorot uniform* distribution.

The initialization of the weight also plays an important part in the design of neural networks. In Glorot and Bengio 2010 the authors show the effects of the activation function behaviour with different distributions for weight initialization. While their experiments were on Dense connected networks and tested in image classification, this paper has supported the Machine Learning community in reaching a consensus as to which initialization-activation pairs work better together.

Generally, when using tanh-activated LSTMs, using *Glorot* initialization is recommended as expressed in Eq. 5.3.

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right] \tag{5.3}$$

This demonstrated an improved performance versus networks initialized with uniformly distributed weights and a sigmoid activation,

$$W \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{5.4}$$

which shows slow and poor convergence. A different experiment in Glorot and Bengio 2010 using softsign activation, $\frac{x}{1+|x|}$, also showed that although the gradient propagation through the layers is characteristically non-linear, it tended to saturate considerably less than the hyperbolic tangent and the gradients flow would result to be more robust.

Since it is a binary classification problem, binary cross-entropy, or log loss, is used as the loss function for all networks. This measures the performance of the model by assigning a probability value output between 0 and 1 when calculating error rates. This is calculated in the following manner:

$$loss = -\sum_{c=1}^{C} y_{o,c}\log(p_{o,c}), \tag{5.5}$$

where C refers to the number of classes, y is the true label for the observed, o, sample and P is the predicted label likelihood. The loss for each class is calculated separately and then summed. In the case for binary classification, the cross-entropy can be calculated as:

	Search Domain
Fixed Settings	
Init	glorot uniform
Epochs	100
Dense Layers *FCN*	[64,32,16]
Searched Settings	
Input Layer *(LSTM & LSTM+ ver2)*	[4,8,16,32,64]
Hidden Layer *(LSTM+ ver2)*	[4,8,16,32,64]
Input Layer *(LSTM+)*	[4,8]
Hidden Layer *(LSTM+)*	[2,3]
Filters *(CNN)*	[32,64,128,256]
Kernel *(CNN)*	[32,64,128,256]
Batch Size	[16, 32, 64]
Optimizers	[rmsprop, adam]

Table 5.6: Table showing the range of values chose for tuning the hyper-parameters.

$$loss = -(y\log(p) + (1-y)\log(1-p)) \tag{5.6}$$

It's important to pay attention to the optimizers for the following reason. In many cases, the Adaptive Moment Estimation (Adam) algorithm (Kingma and Ba 2015) is a preferred optimizer given that it achieves better and faster results than its counterparts Root Mean Square Propagation (RMSProp) and Adaptive Gradient Algorithm (AdaGrad). In Li et al. 2018, visualization of the loss sketch has been said to help clarify how neural networks generalize. The authors also reference works (Chaudhari et al. 2016; Shirish Keskar et al. 2016; Hochreiter and Schmidhuber 1997a) which state that flat minimizers produced with small-batch training generalize better than sharp or spiking minimizers produced by large batch sizes. However, it is mentioned that this is argued against in works (Hoffer et al. 2017; Goyal et al. 2017; Hardt and Ma 2016) which state that generalization is not directly correlated to the curvature of the losses. Regardless, the work by authors Li et al. 2018 on Wide-ResNets shows how flatter losses result in lower error values as opposed to sharp or spiking losses.

A Randomized search on a set of hyper-parameters was performed in where a fixed number of parameter setting combinations are samples from the defined domain ranges. The randomized search is done over number of input nodes, training optimizer and training batch size. Having established a fixed sequence size, the parameter search is run for a generation of 10 models using a 5-fold cross validation training. This means that 5 sets of metrics and statistics are obtained per model to evaluate its performance.

Table 5.6 shows a summary of the fixed set and randomly searched parameters along with the domain range. The randomized sample output for each network set can be found in Annex A.1.

5.7 Results

While the number of options for parameters to tune does not generate an exhaustive combination of options like in Grid Parameter Search, the Random Search it provides an ample range for the selection of optimal values. The parametrization effects on accuracy and metrics are observed. These hyper-parameter configurations are referred to as *Search n*, *n* being the value combination identifier yielded by the Random Search. Setting a fixed seed allows for reproducible results and allows for the same combination of parameters to be used for the architectures to make a fair observation of the effects. The train and validation loss convergence plots are also observed and are included in the Annex.

The analysis consists in comparing the classification statistics of the candidate models, i.e. the ten different parameter combinations, accuracies, F1, Precision and Recall scores per class across the validation folds. The Precision of a model is its the ability of the model to classify all positive samples correctly. Recall is the ability of the model to catch all the positive samples without misclassification. The F1 score is the weighted harmonic mean of the Precision and Recall, where the best value is at 1 and worst score at 0.

The candidates are selected by studying their performance in the different class balancing settings,i.e. balanced, proportional or no weight settings, across the cross validations. The models with the most stability, i.e. those with small variability or spread and high median accuracy and high or acceptable scores, are considered good candidates.

LSTM. The first evaluation is done on the most simple network. The boxplot in Figure 5.5 show the performance accuracy in the 5-fold cross-validated training for this network. It is observed that not using a balancing setting for these experiments consistently achieve good performance. The balancing settings have difficulty achieving good performance and have wide spread of accuracies across the cross validation folds.

In the case of the balanced-weighted experiment, three candidates standout, *Search 4, 5* and *Search 8* as their medians all reach above 80% accuracy in at least two of the three balancing experiments. The other metrics, however, are quite poor. From Figure 5.6, the most stable metrics also point to the *no weights* balancing setting. The model that qualitatively showed the least spread across all balancing settings is model was *Search 5*. Its for this reason that in these experiments is important to observe the class F1, Precision and Recall scores. For instance in *Search 4*, where F1 score for the spiral class, despite having a high accuracy of 86.11% it does not achieve as good performance on the class specific metrics. *Search 3, 4,* and *5* achieved metrics with the least variability above 90%.

LSTM-FCN. The next set of architectures are then evaluated to further continue with the evaluation and selection of the network. This stacked network is evaluated next in the same manner. In this experiment, the addition of FCN showed improvement in Cross-Validation (CV) accuracies particularly in the *no weights* balancing setting. These all reach accuracy

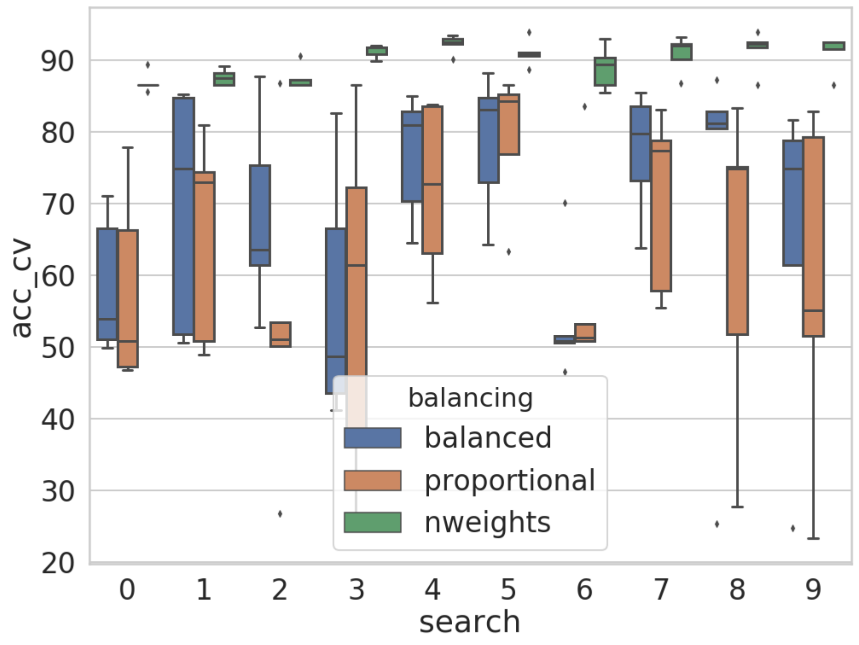
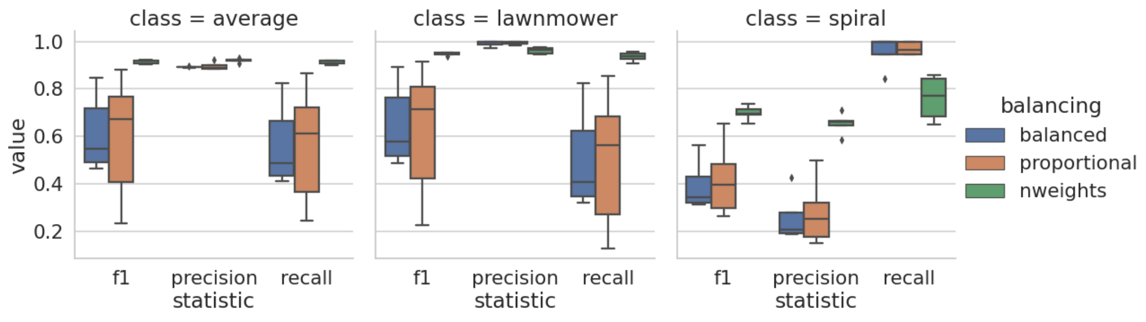


Figure 5.5: Comparison of Random Search experiments in LSTM network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

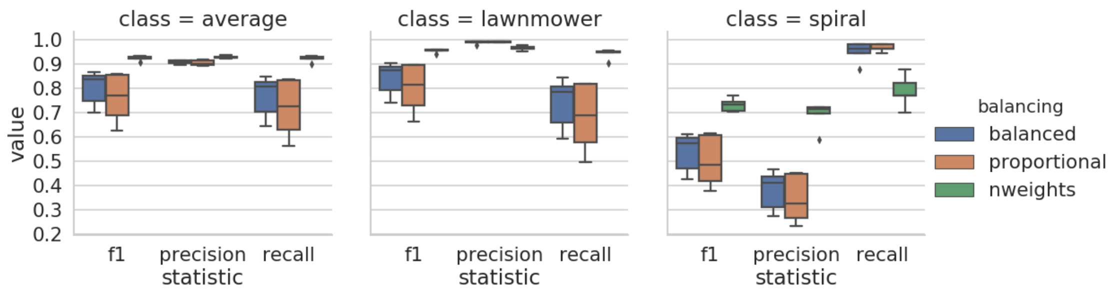
scores above 90% in this setting as shown in Figure 5.7. For the other settings, *Search 6* and *Search 9* are the only models whose has median accuracies are above 80% in all settings. There are other possible candidates are that also could qualify based on their accuracy medians. Upon revision, only configurations 0, 5, and 9 show stable results across all classes in the other metrics. These results are shown in Figure 5.8.

LSTM+. For this network architecture, the addition of the LSTM layer requires an additional parameter in the Random Search. Table A.2 shows the parameters for each candidate model. Figure 5.9 shows how the same balancing setting maintains higher accuracy scores across all configurations. Only configurations *Search 0*, *Search 4* and *Search 6* through *Search 9* were only able to achieve $>$ 90% accuracy. Figure 5.10 shows, however, that most of these showed large variance in the other metrics. *Search 4* and *Search 6* present stable class-specific CV scores across in the *no weights* settings. The scores of other candidates are included in Annex 5.10.

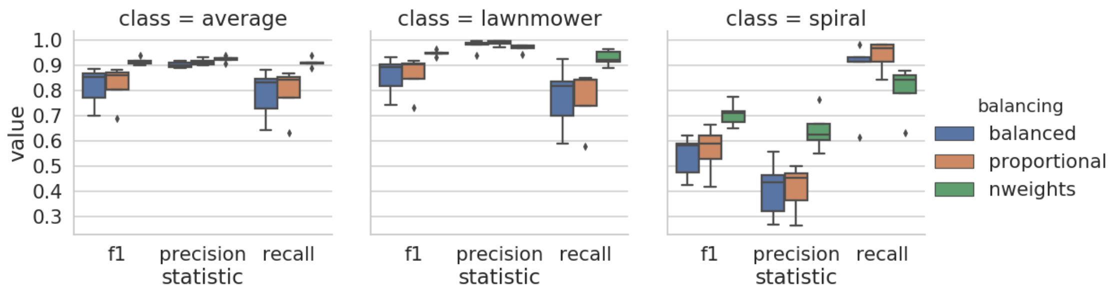
LSTM+-FCN. The additional Dense layers these networks showed little improvement and wider accuracy variances although this could be attributed to the number of nodes in the LSTM networks since the layer size sampling domain is small-valued compared to the 1-layer LSTM sets. This can be supported by observing the classification reports where, even for the non-weighted, the scores are not promising. Only experiment *Search 2* has two



(a) Search 3



(b) Search 4



(c) Search 5

Figure 5.6: Class metrics for class balancing settings on weighted and non-weighed training in LSTM network best candidate searches: (a) Search 3, (b) Search 4 and (c) Search 5. These plots display cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting.

balancing settings that achieve accuracy medians above 80% accuracy, however, their other metrics have large variability. Experiments *Search 4*, *Search 6* and *Search 7* achieve the most stable statistics. However, they present very low scoring in one CV fold.

Its interesting how the lack of class balancing does not deter the model from obtaining acceptable results. So it might be that in this case, where the predominant class surpasses by 630 samples or 73%, is not critical and class balancing might just be required for problems where the lack of samples for a particular is astronomic. Furthermore, the increment of dense layers to the architectures also exhibited no improvement in the models. The following network sets are composed of stacked LSTMs with CNNs networks that will not include the additional Dense layers. The selection of the candidates will continue along the

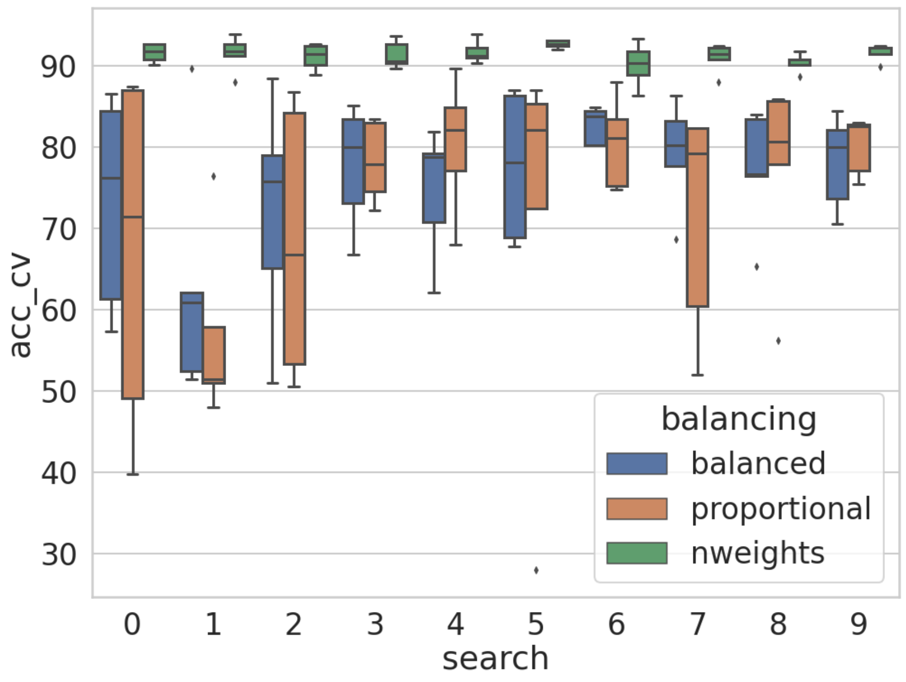
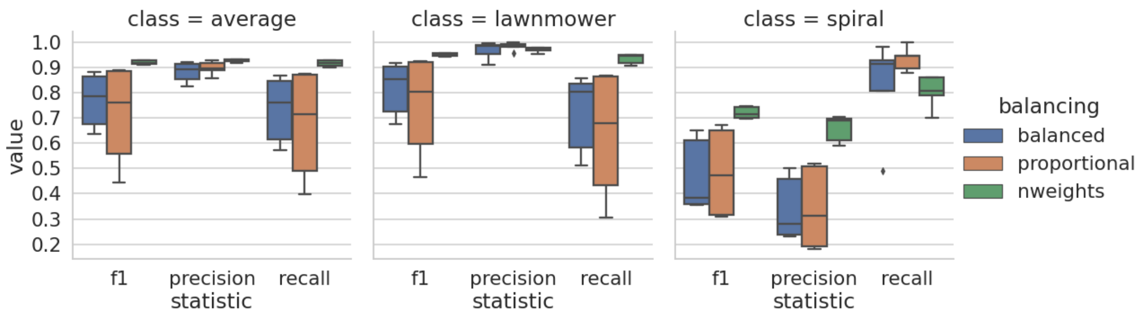


Figure 5.7: Comparison of Random Search experiments in LSTM-FCN network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

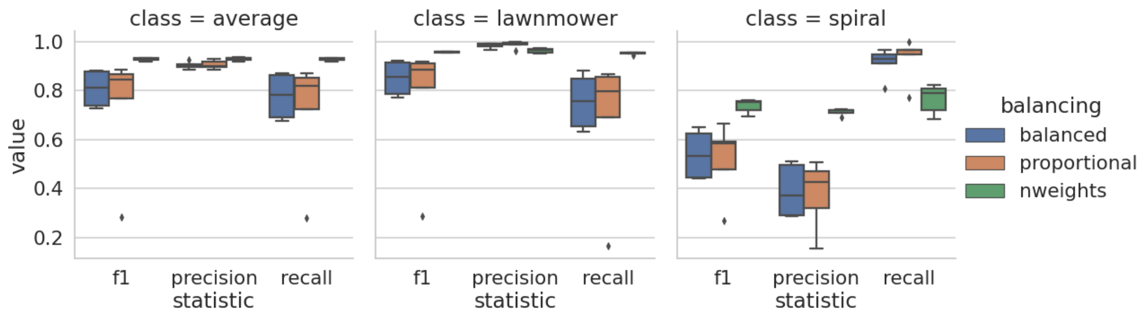


(a) Search 0

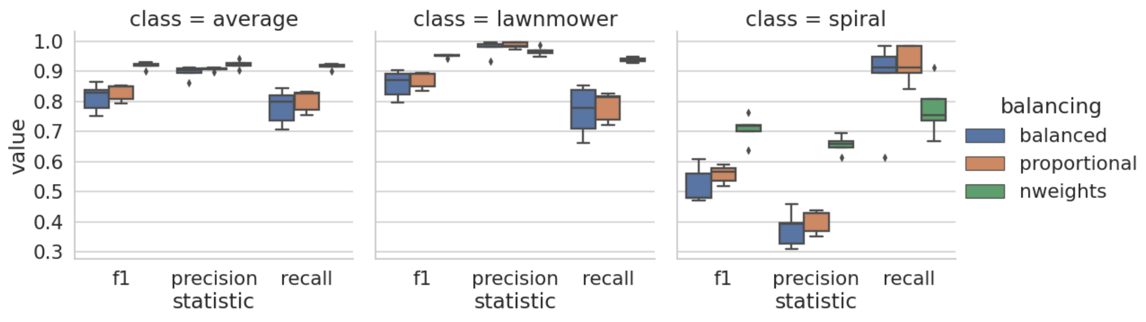
Figure 5.8: Class metrics for class balancing settings on weighted and non-weighed training in LSTM-FCN network best candidate searches: (a) Search 0, (b) Search 5 and (c) Search 9. These plots display cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting. *(cont.)*

same praxis.

CNN-LSTM. The 1-D pooling does not significantly improves the accuracy results compared to the previous LSTM networks. The balancing settings appear consistent here, with the *no weights* balancing setting outperforming the other two settings. Observing the classification report statistics for models with accuracy spreads above or near 90%, only *Search 2, 6* and *7* achieve stable metrics.



(b) Search 5



(c) Search 9

Figure 5.8: Class metrics for class balancing settings on weighted and non-weighed training in the LSTM network.

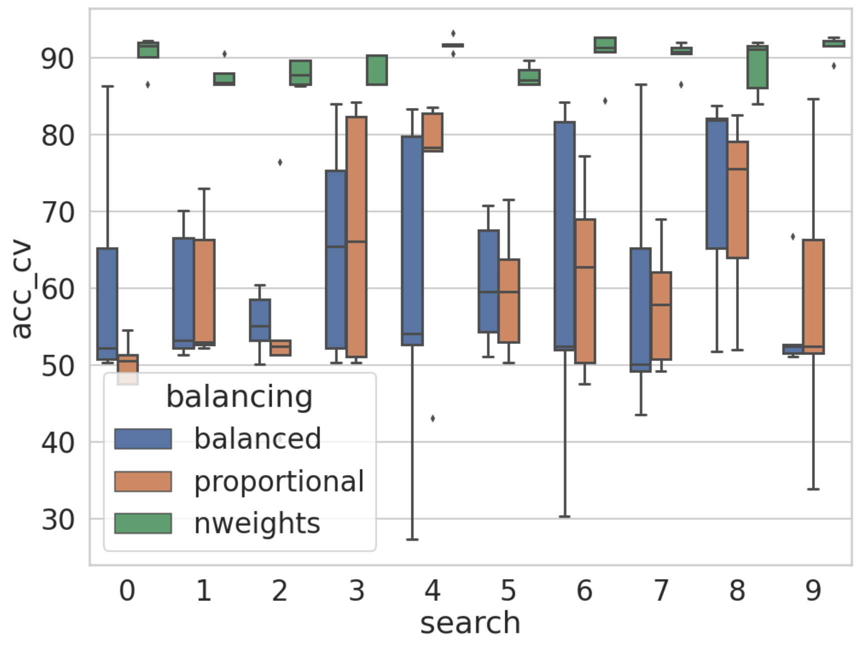
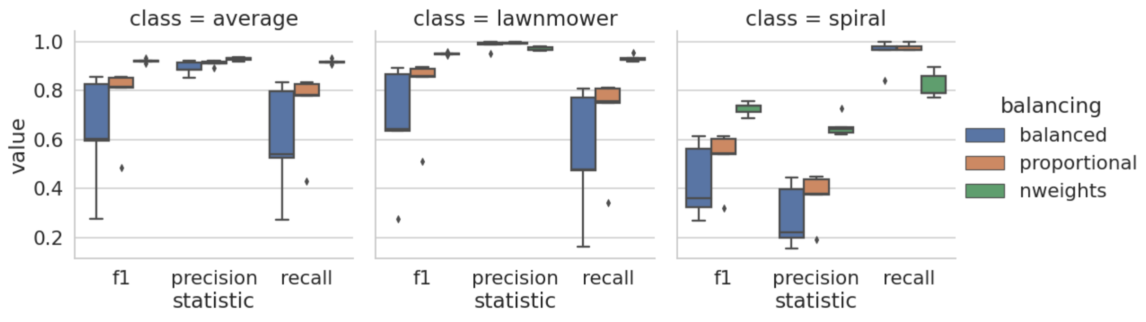
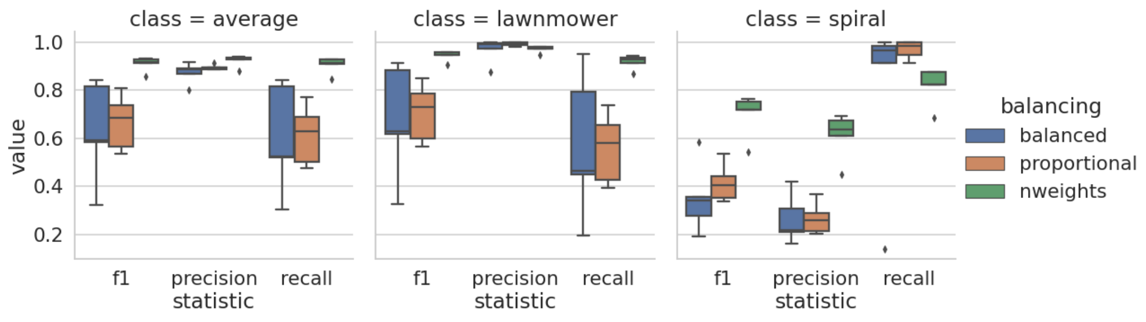


Figure 5.9: Comparison of Random Search experiments in LSTM+ network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

CNN-LSTM+. In this experiment, the additional LSTM layer seemed to reduce the accuracy variability for the *no weight* balancing setting. Like in the previous example,



(a) Search 4



(b) Search 6

Figure 5.10: Class metrics for class balancing settings on weighted and non-weighed training in the LSTM+ network best candidate searches: (a) Search 4 and (b) Search 6. These plots display cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting.

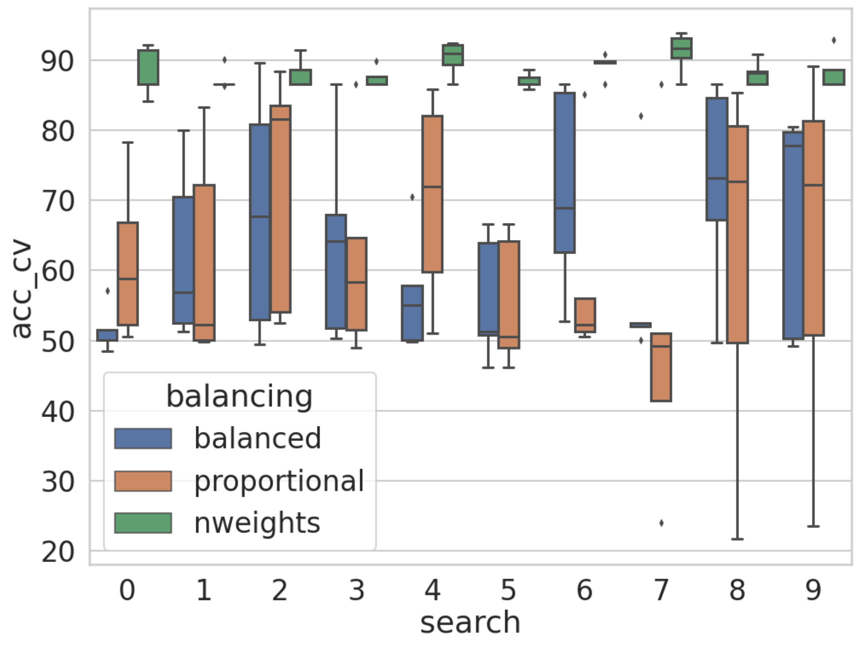
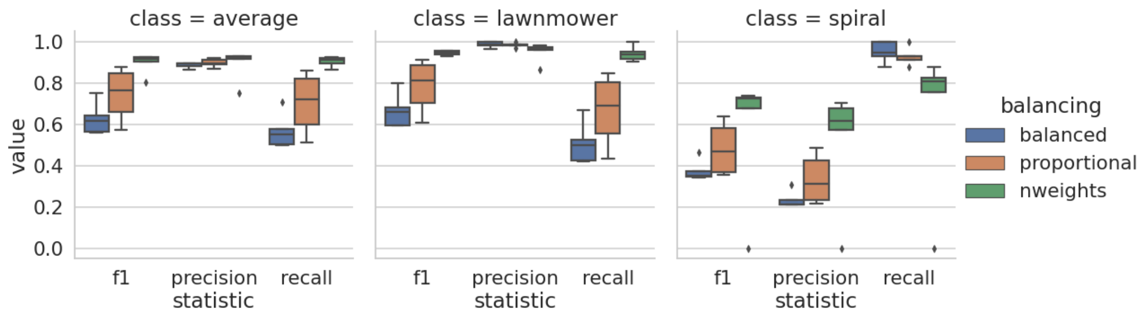
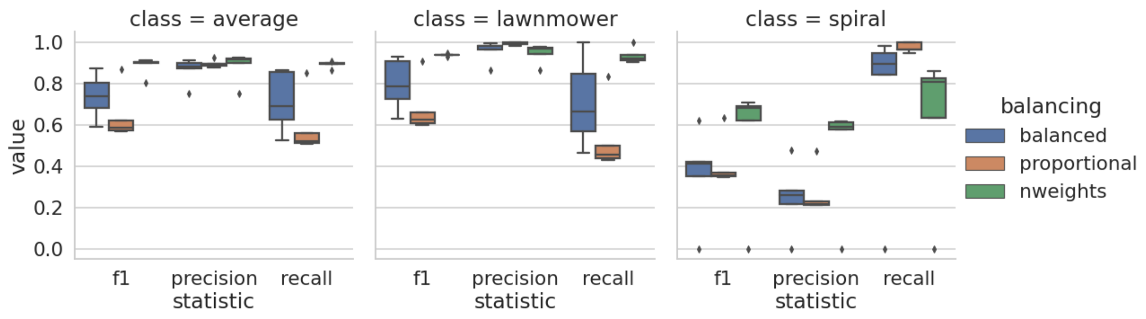


Figure 5.11: Comparison of Random Search experiments in LSTM+-FCN network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.



(a) Search 4



(b) Search 6

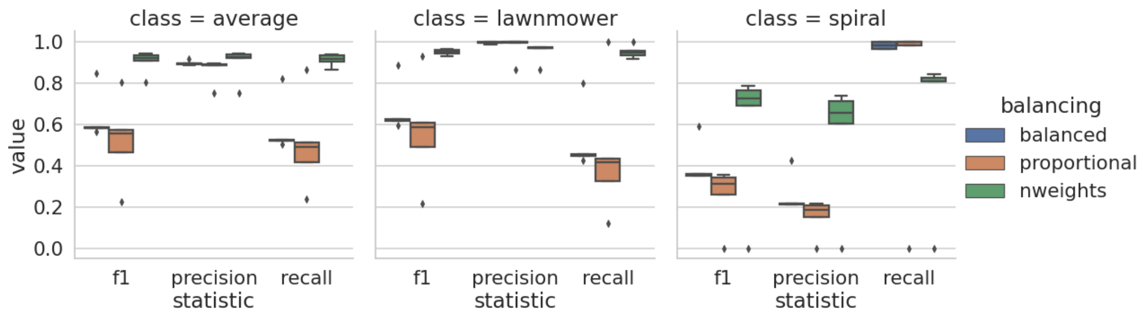
Figure 5.12: Class metrics for class balancing settings on weighted and non-weighed training in the LSTM+-FCN network best candidate searches: (a) Search 4, (b) Search 6 and (c) Search 7. These plots display cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting. *(cont.)*

the configurations whose spreads achieved above or near 90% accuracy were observed. These were *Search 0, 1, 5* and *Search 6*. From these, only *Search 5* showed the most stable classification metrics.

Because the sampling search domain for the Random Search in the CNN-LSTM+ was narrowed, a second version of the CNN-LSTM+ was evaluated with the same search domain wherein the layer size domain range is set as in the CNN-LSTM. The reason for this domain search cutback was to compensate for the increase in number of training parameters given the additional hidden layer in the LSTM. The insight that comes with these last results from the CNN-LSTM is that perhaps the search domain is too narrow to yield contrasting results. The next experiment is therefore identified as version 2 of CNN-LSTM.

CNN-LSTM+ *ver2*. The increase in search space indeed yielded more configurations with better accuracies. Candidates with accuracy spreads above or close to 90% are selected for evaluation on classification metrics. *Search 0, 2, 4, 5, 7, 8* and *9* achieve this, however only candidate *4* has the most stable spread.

From these experiments it was observed that the addition of Dense layers as a FCN network did not necessarily increase the performance on the networks overall apart from



(c) Search 7

Figure 5.12: Class metrics for class balancing settings on weighted and non-weighed training in the LSTM+-FCN network.

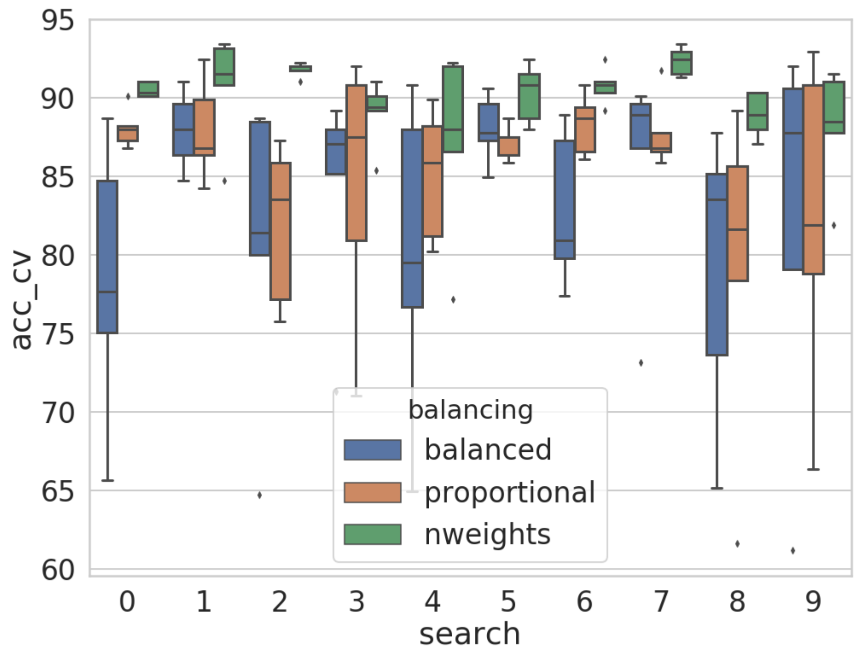
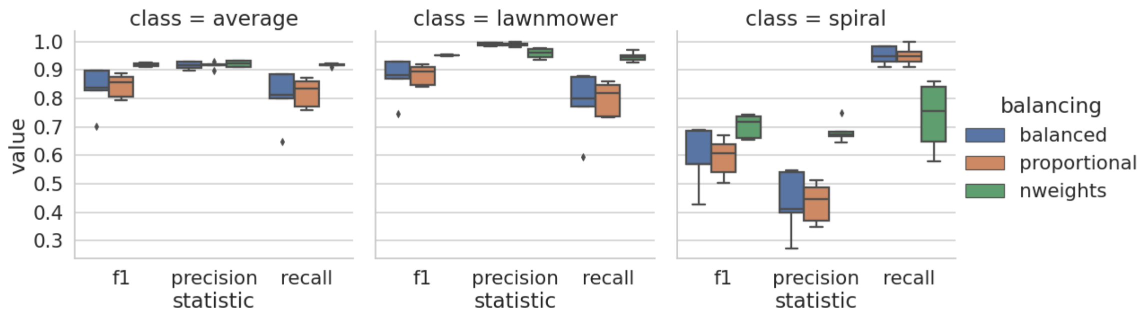


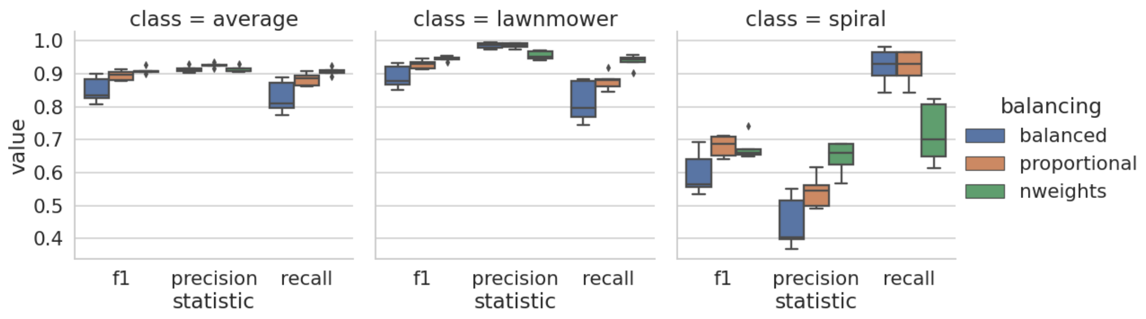
Figure 5.13: Comparison of Random Search experiments in CNN-LSTM network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

the LSTM-FCN. The same can be said about CNN layers. The stacked FCNs in the first experiments by definition disqualify these networks from being shallow, it was deemed relevant to include these evaluations and show that deep is not always conducive to good performance, particularly when dealing with small datasets. The one aspect that did show to convincingly determine good performance was avoiding setting any class balancing. An interesting observation is that *Search 4* was the configuration that was most present in the LSTM experiments. The CNN candidates did not share a common configuration for good scores.

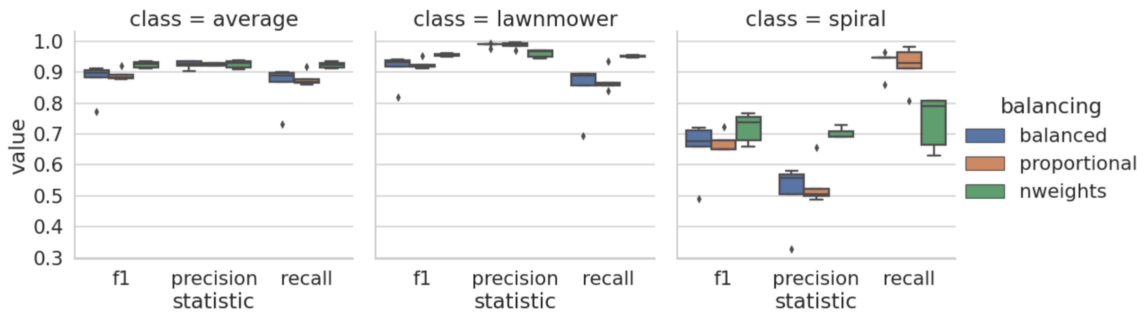
These candidates are subsequently trained without cross-validation sets in all balancing configurations to assert that in fact not setting balancing weights yields the best scores.



(a) Search 2



(b) Search 6



(c) Search 7

Figure 5.14: Class metrics for class balancing settings on weighted and non-weighed training in the CNN-LSTM network best candidate searches: (a) Search 2, (b) Search 6 and (c) Search 7. These plots display cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting.

Their performance is recorded and displayed by F1 score pertaining to the underrepresented spiral trajectory class, *F1 c1*, and *Test accuracy*. Figure 5.19 shows the performance relationship where it can be observed which candidates are both accurate and robust to class sensitivity.

It is observed that the majority of non-weighted models achieve higher than 90% accuracy. No candidates achieve an F1 score higher than 0.8. Only non-weighted candidates surpass the .7 mark with the exception of three other experiments with different balancing settings. Table 5.7 summarizes the configuration hyper-parameter values for the network candidates that satisfy the class sensitivity robustness and accuracy thresholds.

The balanced and proportional weighted models are comparable the lowest performing model with no class balancing. The best performing candidate corresponds to a

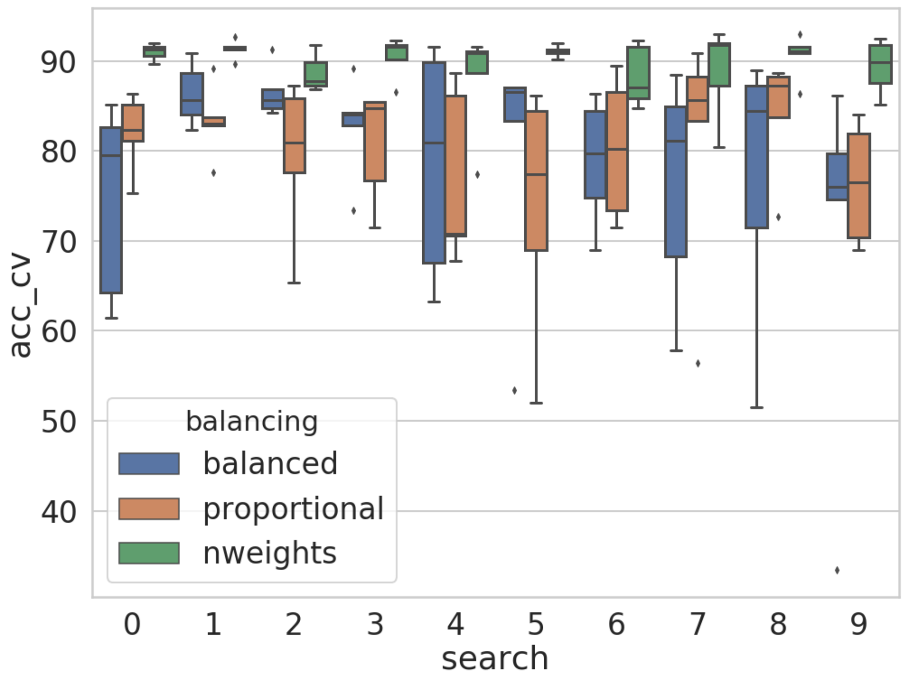


Figure 5.15: Comparison of Random Search experiments in CNN-LSTM+ network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

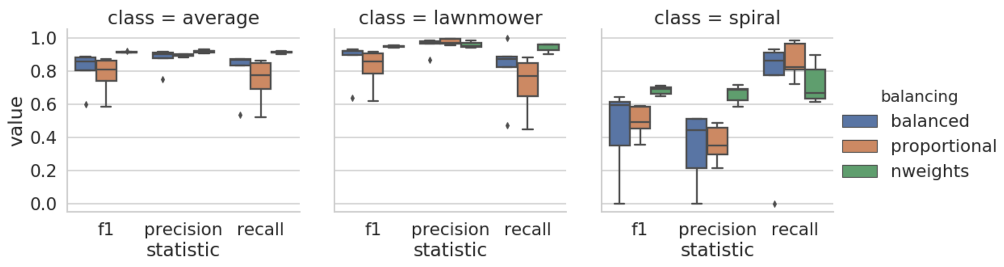


Figure 5.16: Class metrics for class balancing settings on weighted and non-weighed training in the CNN-LSTM+ network best candidate search: Search 5. This plot displays cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting.

CNN-LSTM network. Based on this summary, the discriminative process based on cross-validation variance stability overlooked important high performance candidates as observed by the non-cross validation evaluation.

To further understand how the hyper-parametrization, the following section approaches the explainability capacities of SHAP to understand what fundamentals elements in the parametrization play an important part in the model ability to accurately predict a navigation trajectory. It is hoped that with this explanation the model selection will be done in a more informed and trustworthy manner where the assessment of each model and its hyper-parameters is evaluated from another perspective, in addition to the essential

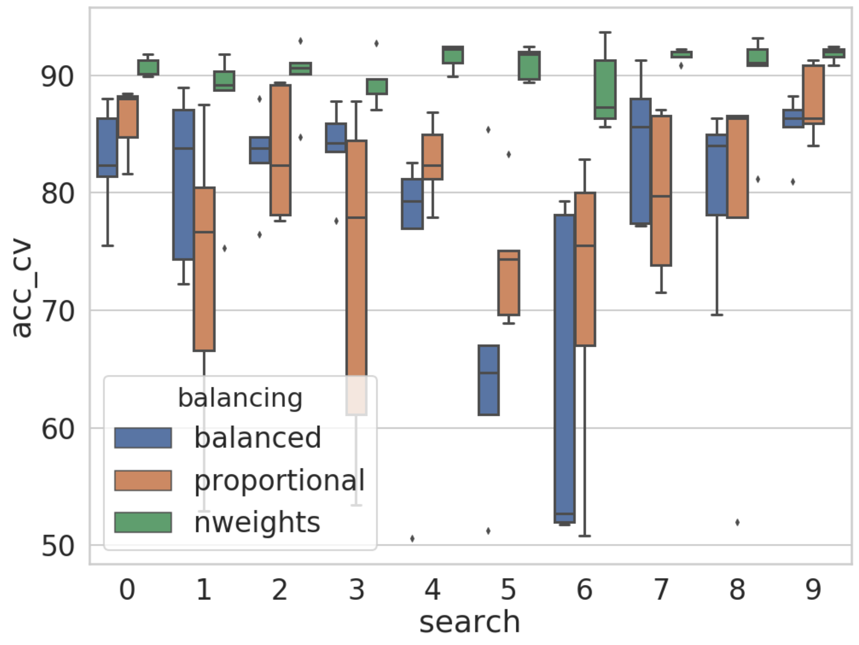


Figure 5.17: Comparison of Random Search experiments in CNN-LSTM+2 network. Cross-validation test accuracies are displayed for each candidate search for each class balancing setting.

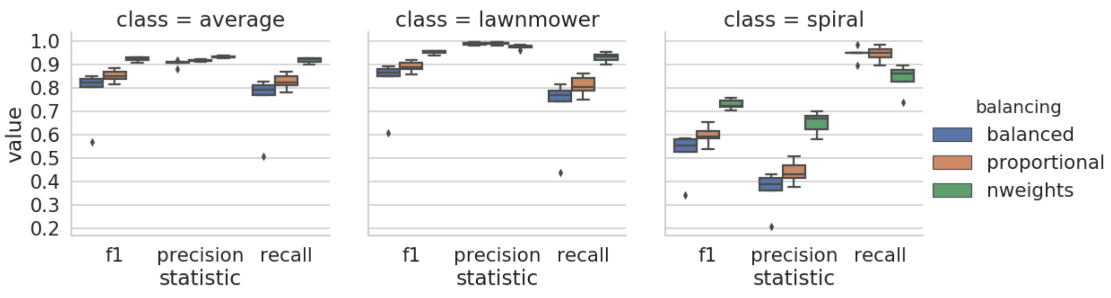


Figure 5.18: Class metrics for class balancing settings on weighted and non-weighed training in the CNN-LSTM+2 network best candidate: Search 4. This plot displays cross-validation scores categorized by class and class average. Each score is displayed according to class balancing setting.

performance metrics.

5.8 Model Selection with SHAP

In the interest of understanding how the hyper-parametrization yields specific results and to trust in the selection of a robust model for this problem, this section presents an additional stage of evaluation of all the models trained without cross-validation sets.

Recalling Section 3.2, Shapley values are used to calculate the contributions of the input features to the models output. SHAP are a consolidation of Shapley values adapted with

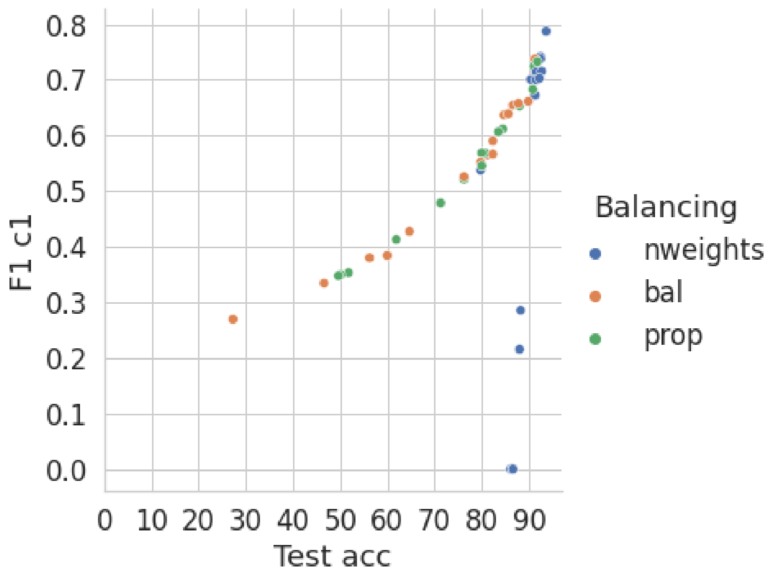


Figure 5.19: Candidates Balancing Evaluation performance in *Test Accuracy* and *F1 score* in the underrepresented spiral trajectory class. Differences in balancing settings are noted in blue for no balancing setting, orange for balanced weights, and green for proportional weights.

Network	Conf ID	Weights	$Acc_{Test}(\%)$	$F1_w$	$F1$ Lwn	$F1$ Spr
LSTM	3	*none*	91.53	0.9204	0.9497	0.7313
LSTM	5	prop	91.06	0.9169	0.9466	0.7246
LSTM+	6	*none*	91.53	0.9186	0.9503	0.7143
LSTM-FCN	0	*none*	92.47	0.9272	0.9559	0.7419
LSTM-FCN	5	*none*	92.47	0.9268	0.956	0.7377
LSTM-FCN	9	*none*	90.35	0.9101	0.9425	0.7007
LSTM+-FCN	7	*none*	92.71	0.9256	0.9582	0.7156
CNN-LSTM	2	*none*	**93.65**	0.9392	0.9627	**0.7874**
CNN-LSTM	6	*none*	91.53	0.9171	0.9507	0.7
CNN-LSTM	7	bal	91.29	0.9196	0.9478	0.7376
CNN-LSTM	7	prop	91.76	0.922	0.9513	0.7328
CNN-LSTM+	5	*none*	92.24	0.9215	0.9553	0.7027

Table 5.7: Best candidates summary of performance metrics.

an additive attribution property. The sum of the SHAP values of all the input features in a sample equals the true prediction (See Eq. 3.11). Hence, it is useful for explaining individual predictions. This method is generally used for interpreting any machine learning model prediction. In this case, the parametrization features and desired scoring outputs are subsumed into their own model to better interpret how the values of such parameters can lead to better model selection.

First a dataset is constructed based on the results obtained from all the experiments in the *no class weights* balancing setting to train a neural network explainer model. In this dataset the input features will be composed by the hyper-parametrization values. These are

LSTM units at the first and second layer, if it exists, number of *Dense* layers, *Batch Size*, number of *CNN kernels* and size of *CNN filter* if it exists, *Test accuracy* and *F1* for the spiral class. These last two features are defined as the output and the rest are defined as input. While *Batch Size* is not used for on-line evaluation or on the test set, it is included as a feature in this problem to determine how it could affect the performance of the network, in particular with the LSTMs, since these are not stateful, i.e. the state is reset after each batch iteration.

The NN explainer model will learn the hyper-parametrization values of a candidate network and predict the real accuracy and F1 score corresponding to the *spiral* class. This model will be used to explain why it makes different predictions for different candidates paying specific attention to the candidate robustness and classification bias. The candidate is deemed robust if a test accuracy of $\geq 90\%$ is obtained. The candidate is deemed class impartial if it can achieve a score of ≥ 0.7 in the under-represented class.

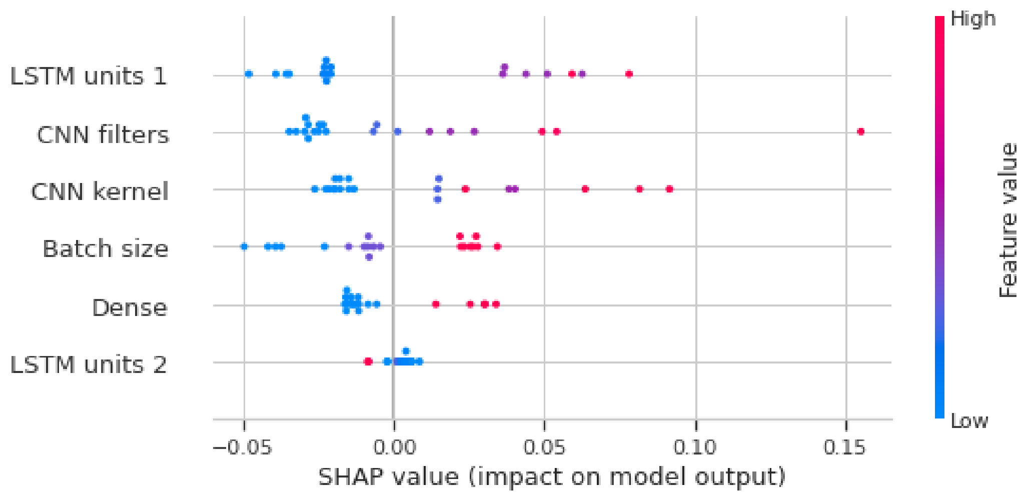
The Neural Network is framed as a regressor with two Dense layers of with sizes, $M = l_1 : 8, l_2 : 2$, using ReLU activation and Mean Squared Error (MSE) loss. One subset of the data is used to train the model for each target output, SHAP values are calculated to find global and local explanations for the hyper-parameters. To understand the contributions of the input features on each output feature, plots in Figure 5.20 displays these for each of the output features. This summary of SHAP values shows similar results for each output feature which means that the features contribute almost in an equal manner to each feature of the output. When the feature values are low it can be observed that they negatively impact the output. Inversely when the features adopt higher values. The hyper-parameters that contribute more significantly to the output are the number of LSTM units in the first layer and the number of CNN filters.

To observe the contribution of hyper-parameter values to the *Test accuracy* and *F1* score in the spiral trajectory class. A few experiments are selected to observe hyper-parameter value contributions to good and bad candidates.

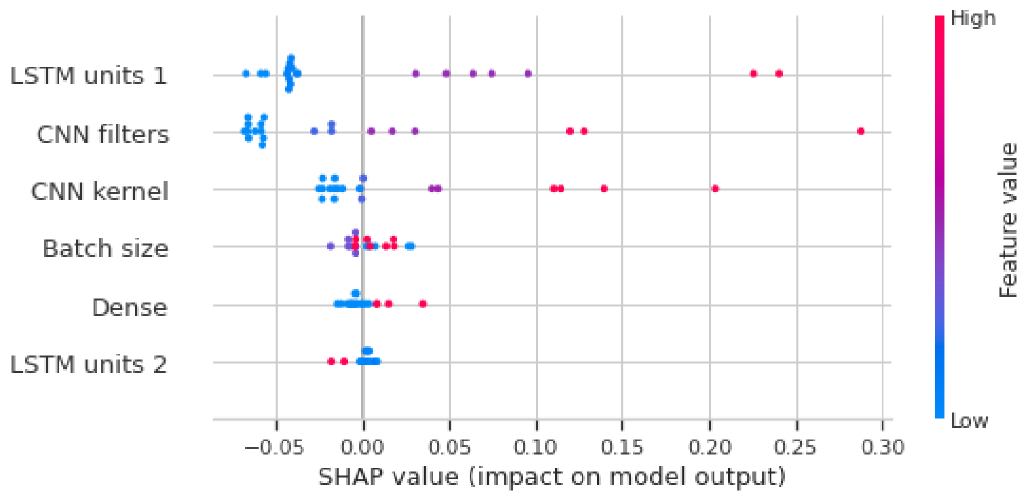
5.8.1 CNN-LSTM Explanations

Starting by contrasting a bad performing CNN-LSTM experiment with a good one, Figure 5.21 shows the single prediction explanation for a CNN-LSTM models with low performing metrics of $CNN - LSTM_0 = Acc: 89.65, F1_{cl1} : .4211$. From this plot the search hyper-parametrization values are displayed for *Search 0*.

Observe the output and base values $f(x)$ and $E[f(x)]$. The base value $E[f(x)]$ according to Lundberg, Allen, and Lee 2017 is the explainer model output, i.e. the regressor model, if there is no knowledge of any features for the current output, in other words, the mean prediction. The different hyper-parameter values is what pushes the explainer model to the actual result $f(x)$. From these observations its shown how a larger number of filters and kernel size contributes to a higher accuracy prediction. The numbers of*CNN filters*,



(a) Test Accuracy



(b) F1 - Spiral Class

Figure 5.20: SHAP values contribution summary for each of the output features. This summary shows that the feature values affect almost identically each feature of the output. When the feature values are low it can be observed that they negatively impact the output. Inversely when the features adopt higher values. The hyper-parameters that contribute more significantly to the output are the number of nodes in the first LSTM layer, the size of the CNN kernel and number of CNN filters.

size of the *CNN kernel* and *Batch size* play an important role in the outcome by positive contributions to a bad performing model.

Let us consider now the same network on configuration *Search 8* and *Search 2* as shown in Figures 5.22 and 5.23 both good performing models. Here it is observed how a smaller batch size negatively contributes to the result, but the similar outcomes are achieved with different number of *LSTM units* and size of *CNN kernel*.

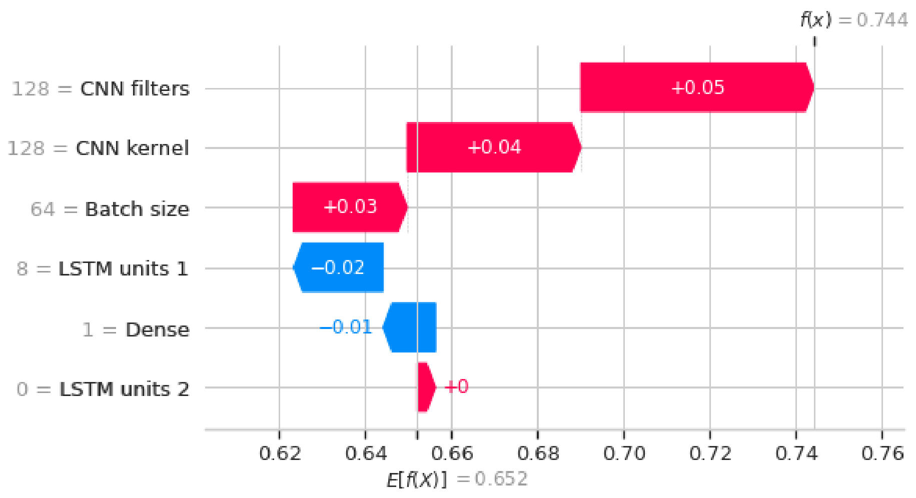


Figure 5.21: CNN-LSTM individual explanations for one example of bad performance metrics. This corresponds to model $CNN - LSTM_0 = Acc: 89.65, F1_{cl1}: 0.4211$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

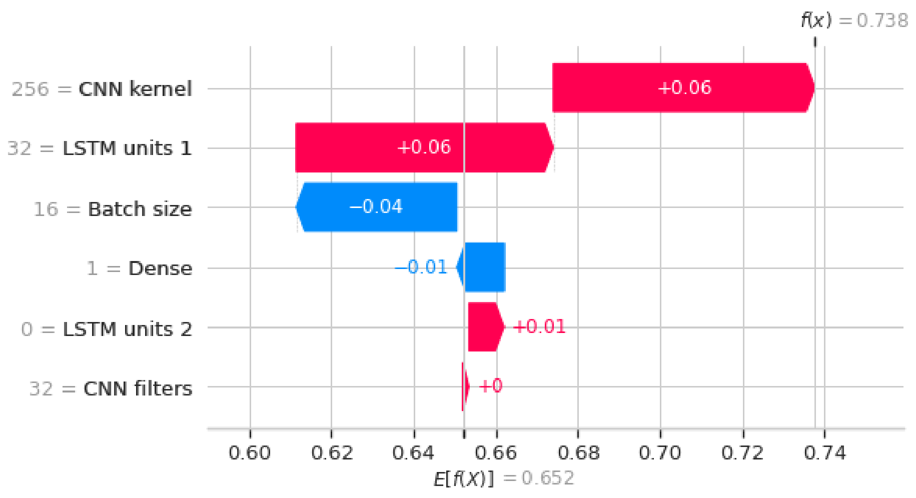


Figure 5.22: CNN-LSTM individual explanations for one example of good performance metrics. This corresponds to model $CNN - LSTM_8 = Acc: 93.88, F1_{cl1}: 0.7833$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

5.8.2 LSTM Explanations

There were also good potential candidates within the LSTM range of experiments. Take for example LSTM-FCN candidates with configurations *Search 4* and *Search 5* with almost equal scores. In the absence of feature extraction CNN layers, the larger number of *LSTM units* and the stacked FCN in *Search 4* (Figure 5.26) pushes the output to a good result. In the case of *Search 5* (Figure 5.25) a larger *Batch size* jointly contributes positively with a reduced number of LSTM units.

Consider now Figure 5.26 corresponding to $LSTM_4$, which is parametrized in the same manner as $LSTM - FCN_4$ except it is not stacked with a FCN network, the lack of which

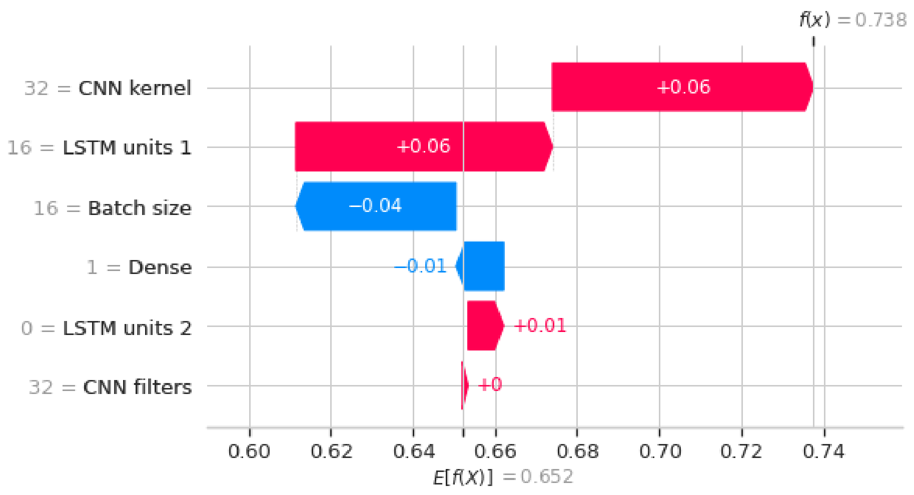


Figure 5.23: CNN-LSTM individual explanations for one example of good performance metrics. This corresponds to model $CNN-LSTM_2 = Acc: 93.64, F1_{cl1}: 0.7874$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

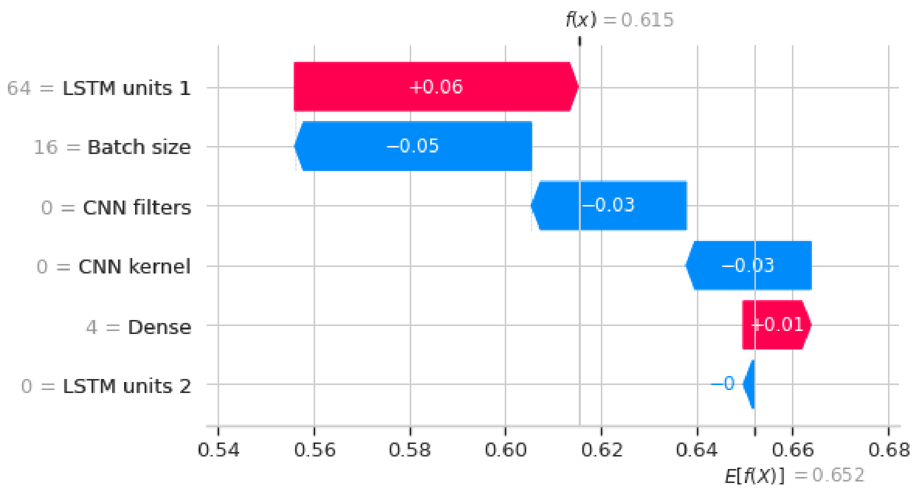


Figure 5.24: LSTM-FCN individual explanations for one example of good performance metrics. This corresponds to model $LSTM-FCN_4 = Acc: 92.47, F1_{cl1}: 0.7333$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

exhibits a lower performance.

From this analysis it was observed that the correlation of the most influential hyper-parameters, *CNN kernel* and *LSTM units 1* interact the most with *Batch size* in where, depending on the large or low values of the influential parameter, the model seems to compensate for a good result or deteriorate in a bad results with *Batch size*. A partial dependency can be obtained and have this effect observed more clearly as shown in Figure 5.27. Partial dependencies show the marginal effect of specific features have on the predicted outcome in a model and would show the type of relationship the target and input features have. Though there are few examples, the relationship shown appears to be positive

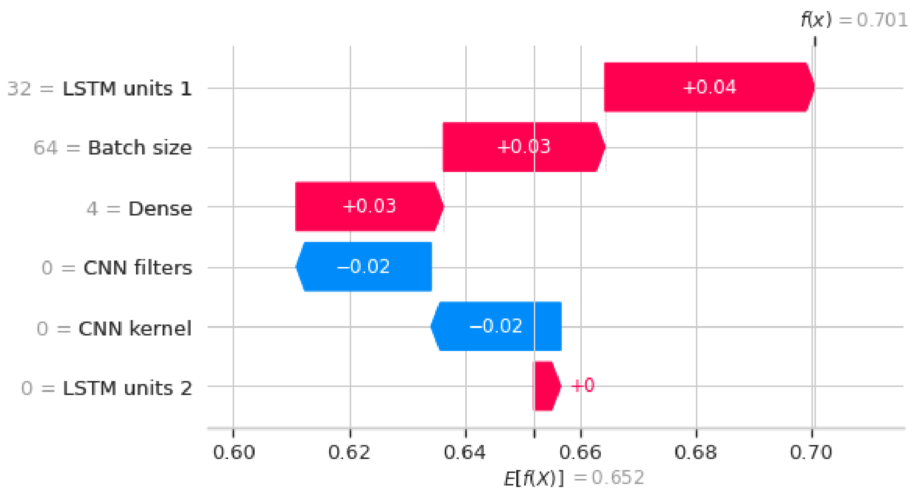


Figure 5.25: LSTM-FCN individual explanations for one example of good performance metrics. This corresponds to model $LSTM-FCN_5 = Acc: 92.47, F1_{cl1}: 0.7377$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

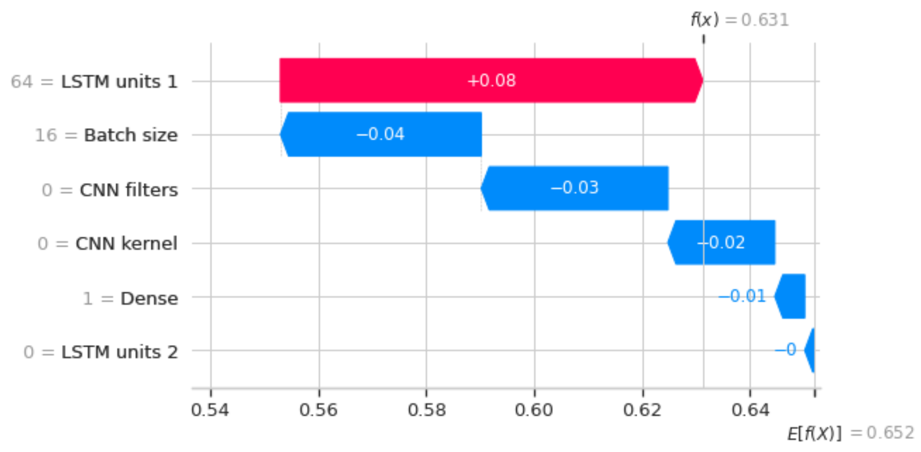
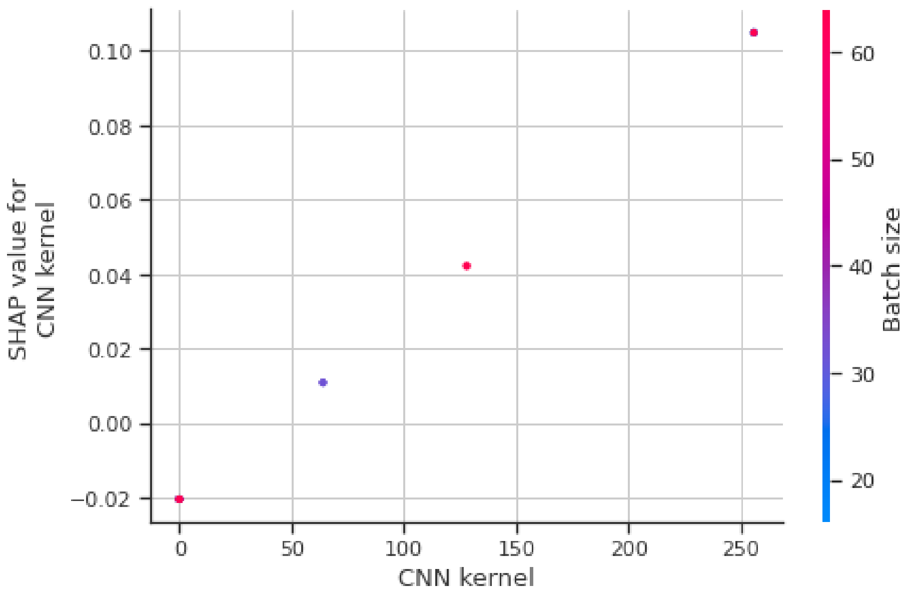


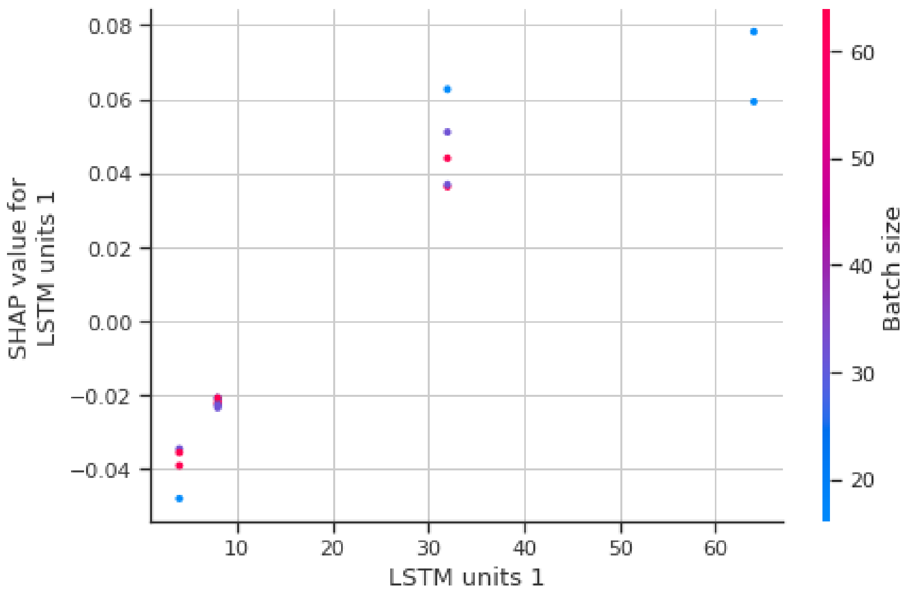
Figure 5.26: LSTM individual explanations for one example of bad performance metrics. This corresponds to model $LSTM_4 = Acc: 79.76, F1_{cl1}: 0.5376$. Bars in red represent positive contributions and bars in blue negative contributions to the final output in the model.

linear.

It is important to point out the SHAP values do not provide causality, however, they are relevant because help explain specific models. By constructing a regression problem scores and hyper-parameters helps understand what drives the candidate outputs to certain values. In this case, they support model selection where there is a need to implement high performing networks with shallow configurations. The individual explanations are useful for model selection as they help understand why each candidate gives a certain prediction accuracy and under-represented class F1 score. In addition, they compare and contrast the impact of each hyper-parameter, especially if transparency in the selection of a shallow network is desired. For this trajectory classification problem both $LSTM-FCN$ candidates



(a) Partial Dependency - CNN kernel



(b) Partial Dependency - LSTM units 1

Figure 5.27: Partial dependencies of the highest contributing features with *batch size*, which is the feature variable they interact the most with. Plot (a) illustrates the partial dependency of *CNN kernel* and *Batch size* pair. Plot (b) illustrates the partial dependency of *LSTM units 1* and *Batch size* pair.

in configurations *Search 4* and *Search 5* and the high scoring $CNN - LSTM$ models are suitable candidates for deployment.

5.9 Summary

This chapter presented a series of experiments for trajectory classification on AUVs. The purpose of these experiments was to compare shallow network candidates and identify the best model parameters. The performance of LSTMs for classifying trajectory patterns executed by autonomous underwater vehicles was evaluated in two manners. The first, by observing the effect of changing the trajectory sequence length and by introducing dropout to the LSTM layers. It was concluded that for this application it might be best to not use Dropout on LSTMs as it decreased the classification accuracy. In addition, the sequence length did not influence greatly the robustness of the model but a fixed window size was selected based on the best performance on the Test set.

The second evaluation was done to observe the effect of class balancing method and hyper-parameter configurations. This analysis entailed explanation assessments for candidate configuration to provide a clear understanding of why the target outputs were achieved and a more informed and transparent selection of deployment models. These conclusions were possible through SHAP explanations, a method for evaluating input feature contributions to target outputs in a specific model.

The implementation of LSTM trajectory classifiers as part of an AUVs AI would be valuable in the setting of field operations as these models are capable of handling raw data and multivariate time-series. Three networks were designed and evaluated against four baseline classifiers: Gaussian HMM, Linear SVM, Decision Tree and Random Forest. An interesting notion arises from this evaluation. It was observed that the the Random Forest outperformed the the best performing NN by 0.02% in F1 and 0.01% in test accuracy. This is not to say that the Random Forest is a better choice of model for this application.

In essence, Random Forests and NNs approach problems by deconstructing them hierarchically or instead of searching for a boundary that splits a specific dataset, like a SVM for instance. However, the Random Forest will split a feature space in a deterministic way whereas the NN will evaluate a feature space and emit a probabilistic observation. This is the main motivation to opting for a NN in this application, since Random Forests are best used when the data is tabular or modeled in ones and zeros. NNs would be best where the data has many intermediate values. In addition, a LSTM maintains the intrinsic sequential nature of the data. Hence, because of the nature of the data NNs is the best choice for this application.

The narrow existing research in trajectory classification for autonomous robots permits the work presented to be a valuable contribution in this field. In addition to producing shallow and robust NNs applied to this problem, an important goal of this work was to demonstrate how this application can support the autonomous monitoring of survey missions. This problem could be further extended to other trajectories such as transitional movements from task to task and sensor data from other subsystems. Part of the work described in this Chapter was presented and included in the proceedings of the 2017 Workshop on Machine Learning for Signal Processing, De Lucas Alvarez, Hastie, et al. 2017.

Chapter 6

Hazard-Oriented Forecast and Detection

"I sense injuries. The data could be called 'pain'."

-T-800, *The Terminator 2*

Time series forecasting is a complex problem in machine learning. Behaviour forecast for autonomous robots behaviours is equally challenging since there are external factors that cannot be accounted for in time series processing and therefore more difficult to model. There are some advantages such as the stationarity of sensor data, where in other real world problems where seasonality and trends are present and the data needs to be made stationary.

Having said that, the occurrence of hazards is an ever present situation for autonomous robots, particularly when deployed in unknown environments. While an exhaustive compendium of the possible dangers an autonomous robot might face is unfeasible, it is possible to equip the system with preventive forecasting routines to best execute its tasks in line with their purpose.

Naturally, the identification of hazards rely on the perception of the robot. Hazard-oriented forecast is different from the two previous problems examined in chapters 4 and 5 where the framing of the problem is defined as a sequence-to-sequence problem and not sequence-to-label. However, LSTMs are versatile and can also be used in time series forecasting.

The approaches explored in this chapter will integrate architectures which combine LSTMs and the Encoder-Decoder concept. The encoding and decoding of time series works by compressing the data via an encoding function and a decoding function for reconstructing the compressed representation. These encoders and decoders are essentially parametric functions that can be trained to minimize the reconstruction loss. In this manner, the Encoder-Decoder model provides a form for any neural network architecture. In the case of the LSTM, the encoder-decoder model provides a pattern for using LSTMs to tackle difficult sequence-to-sequence forecasting problems.

6.1 Application: Mobility Hazard Prediction

Autonomous robots consistently encounter unforeseen dangerous situations during mission execution. Anticipating such events is of utter importance as it can trigger preemptive or preventive actions that minimize damage to the system or any endangerment to the accomplishment of a task. This chapter examines how such hazardous events can be inferred from data. Similar to task modeling, such events are mostly characterized by the sensing mechanisms available.

Time series analysis is certainly relevant for this task. As the rover advances into an area where the condition of the terrain may cause it to slip or tip over the orientation sensors will register signal overshoot. The goal is to forecast the hazard by detecting the behaviour that leads to this overshoot. Like in Chapter 5, time-dependent analysis of the data is required to assess the status of a system and therefore LSTMs are great candidates for capturing the dependencies in time among a series of data streams Hochreiter and Schmidhuber 1997b.

Context During planetary survey missions, exploration rovers need to overcome challenging situations during mobility. Unknown terrain can often pose a threat to the completion of a mission. Steep slopes and loose soil can cause the vehicle to become slip and flip over. Commonly planetary exploration rovers execute trajectories at a conservative pace ingeniously designed to traverse uneven terrain. However, in the event of an unexpected hazard to the rover, the mission can become compromised. Therefore, timely preemptive approaches are crucial for ensuring the integrity of the robot and the success of a mission.

This chapter keeps the general research questions in effect and, in addition, postulates context-specific questions. Having set the context of interest for the Autonomous Vehicle, two goals arise from this problem.

1. How early can the trajectory of the Autonomous Ground Vehicle (AGV) be forecast accurately?
2. How accurately can a mobility hazard be anticipated?

The first question deals with determining the number of input time steps that best predict the next *n* time steps taking into account the frame rate of the sensors. This is referred from here on as the input and output sequence lengths used to train the model. The second question deals fundamentally with an anomaly detection problem, wherein the output of the learned model is used to detect imminent mobility hazards.

The AsguardIV dataset, used in this chapter, consists of orientation sensor data from an AGV and applied for mobility hazards in uneven terrain in a lunar analog setting. The following section describes in detail this dataset.

6.2 AsguardIV Dataset

The name of this dataset comes from the survey rover AsguardIV designed and built at the Robotics Innovation Center (RIC)-German Research Center for Artificial Intelligence (DFKI). This is a collection of survey missions as part of the Environment Modeling and Navigation for Robotic Space-Exploration (Entern) project (Domínguez et al. 2018). The data was collected from the rover performing autonomous and manual navigation in a moon crater analog environment. The crater analog was constructed based on real moon images taken by Apollo missions and multiple of the challenges of the real environment are present, e.g. slopes with different inclinations, rocks and boulders. The rover traverses over surfaces with inclinations between 15° and 35°.

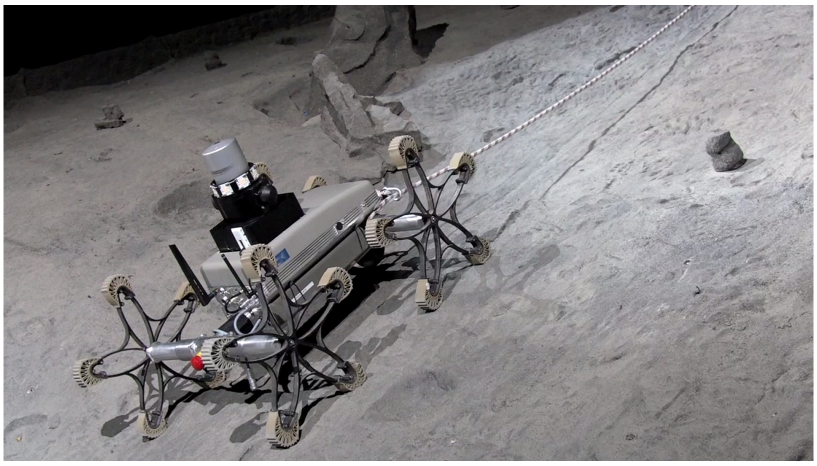


Figure 6.1: The AsguardIV rover in a lunar analog.

The data is a collection of sensor readings from a series of 10 trials that total approximately 250,000 samples. The readings come from the calibrated sensors module of the rover which transmits acceleration and orientation in three axis (x,y,z) of the Inertial Measurement Unit at approximately 100Hz. The data is in log format containing other types of information other than the sensor readings, and thus, it required extraction of the messages that contained only the navigational data. The most relevant message to identity the goal behaviour in this case is the IMU data. This includes calibrated sensor readings conforming a total number of six features:

1. **Accelerometer** Values in numerical float format:Acc_x, Acc_y, Acc_z.
2. **Gyroscope** Values in numerical float format: Gyr_x, Gyr_y, Gyr_z.

Similar to the REGIME dataset, this is a collection of survey missions recorded at the RIC-DFKI space hall where a Lunar surface analog is set up (Figure 6.1). However the application of this dataset is different. The logs are short survey missions in which the AsguardIV rover traverses up and down from a rocky construction analog to lunar soil. The main objective of these recordings was to register hazardous mobility functions such as the rover tipping over while traversing uneven terrain. As such, each trial may or may not contain such anomaly readings and are unlabeled for unsupervised learning. Upon recording these trials, however the following mobility conditions arose:

1. **Normal.** No hazardous motions while traversing terrain.
2. **Tip-over.** When the rover flips over due to uneven and/or steep terrain.
3. **Skid.** When the rover slides in its moving or still tracks due to the uneven and/or steep terrain.

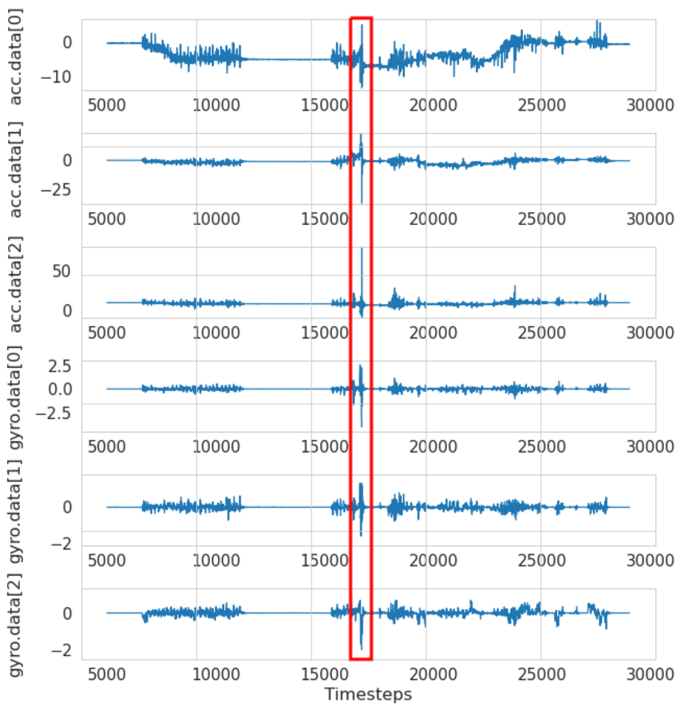
Because the vehicle was being observed on-site it was possible to note for each trial if any of these conditions took place. However if the vehicle is out of sight, the characteristic signal patterns of the rover wheels will not easily characterize all conditions and if so, it is not easy to differentiate by mere visual analysis what kind of hazard it is. An example of is illustrated in Figure 6.2 where one shows a tip-over motion that required a manual stop to avoid damage to the rover. The other could be mistaken as a hazard containing experiment, however there was no actual hazard, but the rover descending too fast for a moment.

The robotic platform is a four star-like wheels skid rover. The star-like wheels are conceived to help the rover traverse challenging surfaces where round wheels would have too much slippage. The particular shape of the wheels produce an undulating profile on the height of the robot even on flat surfaces. The design aims to facilitate navigation in unstructured environments, like craters or caves in other planets or moons. The total collection of sensor readings are from 10 trials from which half was taken for training and the remaining half for evaluation.

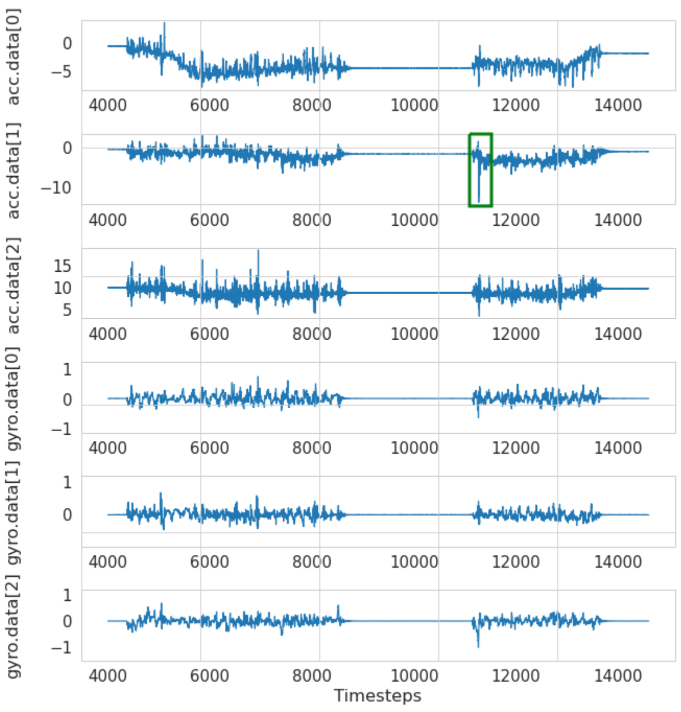
The learning problems that will be tackled using this dataset are of forecasting and anomaly detection. This is a very relevant topic in general in ML but is particularly interesting and valuable for NN-enabled Autonomy for AGVs. The following section presents how the candidate architecture is used for this purpose in more detail.

6.3 Encoder-Decoder LSTM

While LSTMs can be used for time series forecasting, there are some considerations to be made beforehand. A LSTMs alone can be used to model one-step forecasting problems. However, this problem requires for multi-step forecasts, that is, the forecast horizon requires more than one time step. For this the LSTM needs to be contained within the encoder-decoder model. The justification on using this approach concerns the interpretation of the output. If using a standard LSTM network the number of nodes in the output layer needs to match the number of output time steps. This could be done with an LSTM by having the number of nodes in the output be of size $n * m$ for m number of samples with n time steps, but the time steps of each sample would be flattened in the structure of the



(a) Hazard



(b) No Hazard

Figure 6.2: Examples of the AsguardIV performing two missions. These illustrate sensor plots across all axis for the accelerometer and gyroscopes. Figure (a) contains a mobility hazard across all sensors (marked in red). Figure (b) shows a mission with no hazard but some readings of the accelerometer in y could be confused as one (marked in green).

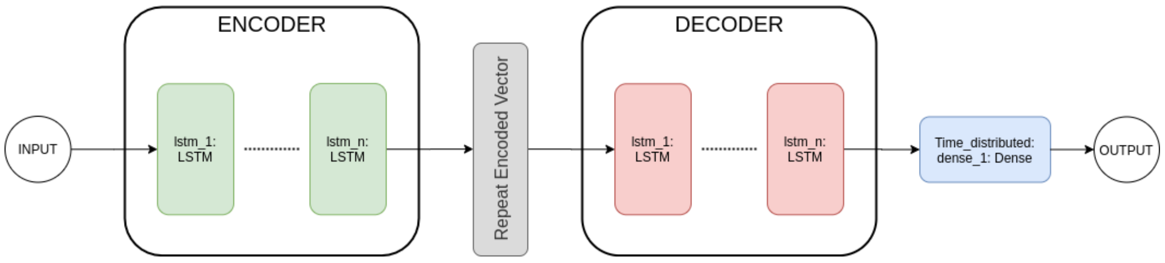


Figure 6.3: Architecture of the Encoder-Decoder LSTM. Combining the Encoder-Decoder model with an LSTMs supports the learning of temporal structures to forecast sequences.

LSTM and each output needs to describe a specific time step for a specific sequence. This is not optimal.

Sequence forecasting is what is fundamentally known as a sequence-to-sequence problem. With a Encoder-Decoder model, it is possible to address this sequence prediction problem by harnessing the time series processing capabilities of the LSTM. As reviewed in Chapter 3 the model is comprised of two integral parts as its name states: the encoder and the decoder. The first learns the relationship between the time steps in the input sequence. LSTMs layers are used for this purpose wherein the output of the encoder is the learned representation of these correlations.

Figure 6.3 shows the general architecture of the Encoder-Decoder LSTM. For this problem hyper-parameter tuning was performed using the HBBO approach. The number of optimal layers was also optimized. An additional factor also included lag in data. This lag could represent any potential jumps in sensor data or intermittent sensor failures.

The sensor readings are transformed into a format that can be used for the forecasting problem. The requirement is that for a given past sequence of readings $X_{t-n},\ldots,X_{t-1}$ the current n time steps $X_t,\ldots,X_{t+n}$ can be predicted. This can be framed into a sequence-to-sequence problem where the first sequence is given as input into a model that learns the second sequence as output.

The input-output sequences are a multivariate vector of six features:

$$X_t = \{accel_t^x, accel_t^y, accel_t^z, gyro_t^x, gyro_t^y, gyro_t^z\}, \tag{6.1}$$

This example applies for all time steps. The superscript being the axis id of each sensor. These experiments were done with time series cross-validation.

6.4 Model Selection

The configuration space for the HBBO optimizer was specified as the number of cells or units in each LSTM layer with a lower boundary of 4 units and an upper boundary of 64 units. Other settings for experimentation outside the HBBO were used for experimentation such as the network depth and sequence length. Table 6.1 list these settings.

HBBO Hyper-parameter	Values
Cells	$\{4, 64\}$
Maximum budget (Epochs)	100
Configuration Settings	**Values**
LSTM Depth	$\{1, 2, 3\}$
Sequence Length (in ts)	$\{25, 50, 100\}$
Lag	$\{0, 10\}$
Dropout	20%
Fixed parameters	**Value**
Loss	MSE
Batches	16
Optimizer	Adam

Table 6.1: Configuration and Hyper-tuning Parameters

The set of parameters yield a set of three HBBO hyper-parameter tuning experiments for a defined network length for each sequence input-output length. This is repeated for three settings: no lag, no lag with dropout and lag of 10 time steps. The readings come from the calibrated sensors module at approximately 100Hz. The input-output length of 100, 50 and 25 therefore represent that the networks are forecasting sequences into 1 second, 500 and 250 milliseconds respectively. This totals to 27 experiments.

Some settings remained fixed for all the experiments. The loss metric was set to MSE over Mean Absolute Error (MAE). Even though both metrics can be used for problems that involve continuous random variables, MSE pays better attention at detecting extreme values. All the networks used the efficient version stochastic gradient descent, *ADAM*, as optimizer.

As previously explained, the Hyperband will increase budget on configurations that show increased improvement at each stage of it. Naturally, these candidates have all been trained on full budget. The results are organized categorically by non-hyper parameter settings. The coefficient of determination, or R^2 score, was used for selecting these candidates. The train, validation and test scores are plotted side by side to observe the learning behaviour of the networks.

At a global level, the most notable observation is that the data fit of the networks is most stable for the lagged input of 100*ms* showing less variance, or less sensitivity, to the data in all sets than the other settings. At a local level, it is relevant to note that some over-fitting occurs the most in the 3-layer LSTM Encoder-Decoder candidates.

By isolating only the R^2 scores for the test set in Figure 6.4 the global behaviour is more noticeable. In addition, more variance is observed for networks with more than one layer in the case of inputs with no lag. These networks consistently show outlier scores in the CV. The time series CV is done by using 70% of one trial and the remaining 30% is used for validation. This is to be expected as the data contains few examples of the rover exhibiting

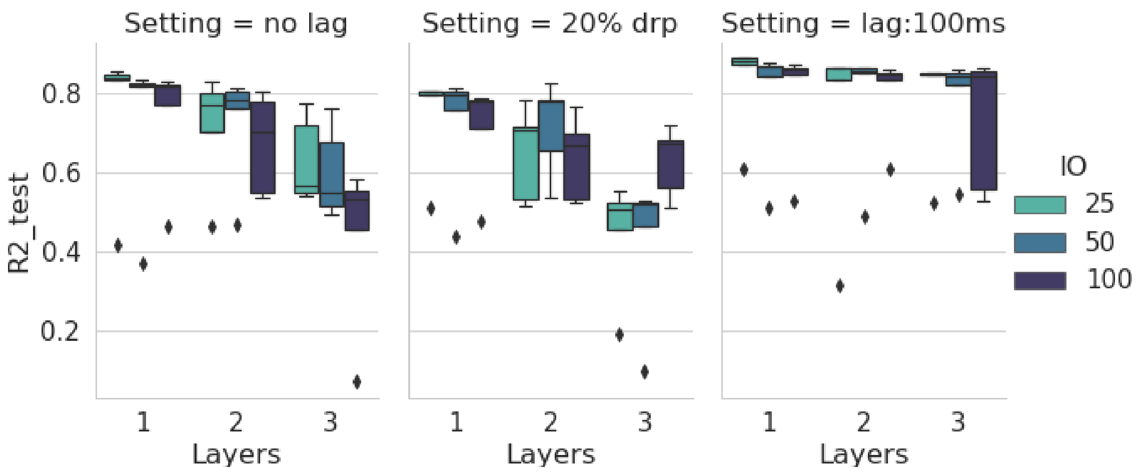


Figure 6.4: Categorical Boxplots showing only R^2 scores for the test set for each Encoder-Decoder depth for a specific non-HBBO setting.

dangerous motions, like sliding or tipping over while ascending or descending. in addition, only half of the data was used for training. Thus, some train sets may not contain hazardous motions.

Some important attributes are noticeable from these preliminary results:

1. Generally, the fewer the layers of an LSTM Encoder-Decoder the best chance it will have of it achieving a score of at least 90%

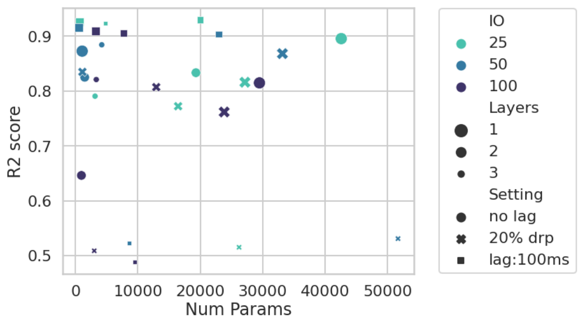
2. However, using a 10ms input lag achieves more stable scores than other settings in smaller networks

3. A 10ms input lag consistently outperforms other settings tests in any network depth.

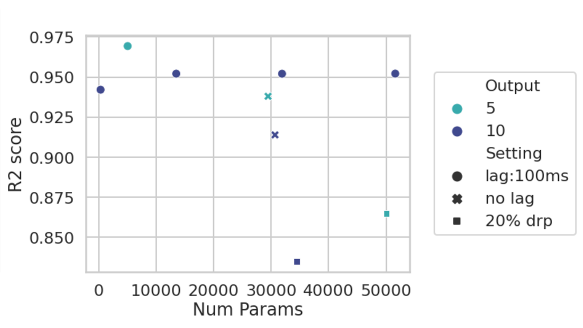
These evaluations struggle to reach above a .9 R^2 score, but training with 70% can improve the CV scores. The choice of using only 50% initially is solely to accelerate accelerate the training time and set up a guideline for the parametrization. With this in mind, all of these networks were re-trained using 70% of the data without time series CV to validate these points.

These scores of the retrained networks are now observed in Figure 6.5. Figure 6.5a shows the scores for all the candidates. This figure shows how an input of 250ms with a 100ms lag input consistently performs better than its counterparts in any layer depth. This is relevant since it may be the case where a particular network may have fewer trainable parameters than another one with a fewer number of layers. Ideally, when embedding these networks in autonomous robots a small robust network with the least amount of trainable parameters possible is desired for energy efficiency purposes.

Because there is no known threshold for ideal energy consumption, all the networks that have achieved higher than a .9 R^2 score are selected for comparison. A total of seven networks qualify and they are summarized in Table 6.2. The common setting for these



(a) For all candidates



(b) For variable output

Figure 6.5: Figure (a) R^2 by number of parameters illustrates the hyper-parametrization behaviour for the candidate networks according to different settings: number of layers, input-output sequence length (IO), presence of dropout, lagged output or no lag. Similarly figure (b) shows the same scoring for variable input-output experiments.

Setting	IO (ts)	Layers	Cells	Params	Test R^2
lag:100ms	25	1	[6]	666	0.925
lag:100ms	25	2	[40, 12]	19990	0.929
lag:100ms	25	3	[18, 5, 5]	4782	0.923
lag:100ms	50	1	[5]	496	0.915
lag:100ms	50	2	[14, 37]	22974	0.903
lag:100ms	100	1	[15]	3276	0.909
lag:100ms	100	2	[21, 11]	7720	0.905

Table 6.2: Settings of selected networks and their test score.

Setting	In (ts)	Out (ts)	Layers	Cells	Params	Test R^2
lag:100ms	25	10	1	[4]	350	0.942
lag:100ms	**25**	**5**	**1**	**[19]**	**5060**	**0.969**
no lag	25	5	1	[48]	29478	0.937
no lag	25	10	1	[49]	30680	0.913

Table 6.3: Settings of selected networks and their test score.

networks is a 100ms lag input. There appears to be no improvement in score despite the varying number of cells and layers among the networks. However, the best performing models have input and output lengths of 250ms.

Having defined this, a second set of experiments that have a shorter output sequence of 5 time steps but maintaining the 25 time step input was performed. Figure 6.6 illustrates the sliding window for data formatting for this setting. Figure 6.5b shows the results the performance results and are summarized in table 6.3. The same initialization parameters from Table 6.1 are used but this time the network depth was added to the HBBO optimization to expedite the process.

This evaluation showed improvement of the first HBBO evaluation by 4%. Some important notions that are taken from these findings help in selecting the deployment

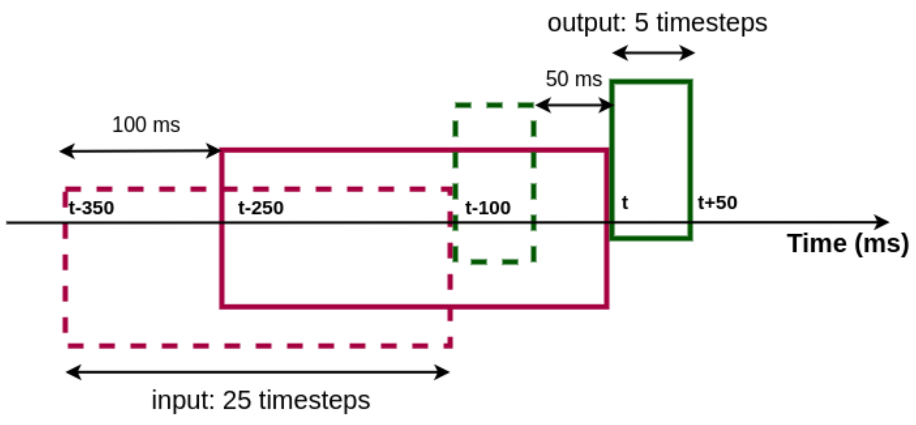


Figure 6.6: Sequence input formatting for best model.

models which also respond to the first question postulated at the beginning of the Chapter: *How early can the trajectory of the AGV be forecast accurately?*. Initially it was speculated that it could be possible for some networks to forecast up to 1 second in advance. However it was observed that the particular motion of the AGV fluctuates substantially due to the wheel design. The models showed that the robustness could be improved by reducing the length of the forecast sensor signal precisely with 50ms ahead of time with a 250ms input.

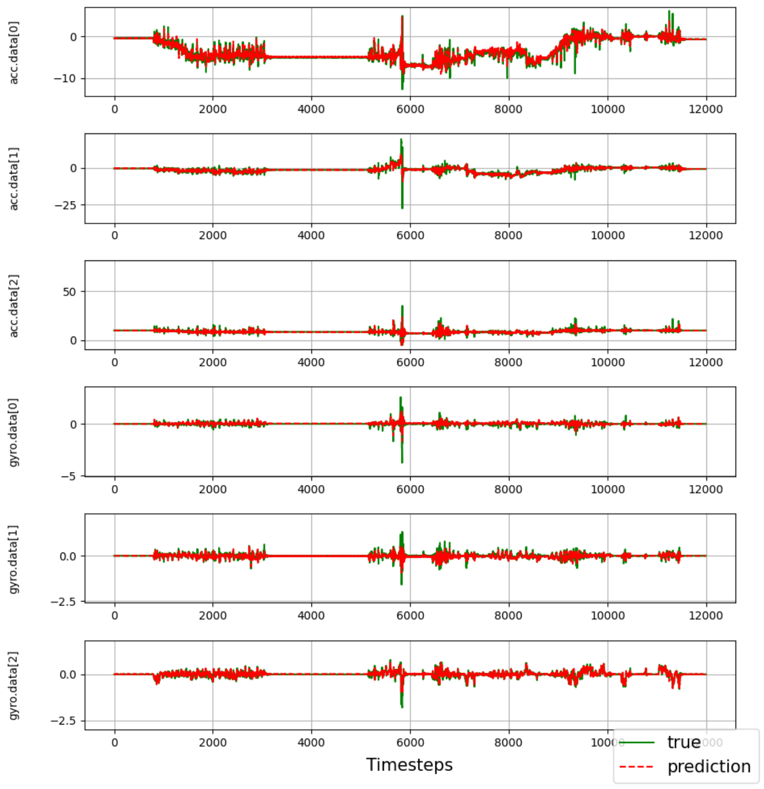
6.5 Hazard Forecast

The selection of the model now allows to evaluate qualitatively the robustness of the forecast. The hazardous conditions that occur in the trials include tip-over and skidding motions. The selected is able to accurately forecast AsguardIV's motions along the lunar analog. Figures in 6.7 illustrate this for each of the test trials. Recall the best network R^2 score of 0.969 with all the training data.

The motion sequences for each of the test trials develop in the following manner. For the first test trial, a goal point was set near the middle of the slope and an autonomous mission solution was generated. Mission started and the first goal point was reached. On the way back, the rover descended hardly and almost rolled down by tipping over. This event can be observed in figure 6.7(a) a bit before $ts = 6,000$. At this point the mission was stopped and the robot was taken down in remote operation mode.

The second test sequence presented no dangerous motions but does include a skidding motion. This sequence was planned as a forward ascent and backward ascent. On the way down, the rover lightly skidded in a crater and finished a smooth backward climb descent. This can be observed near $ts = 3,500$.

The third test sequence repeats the same plan starting from a different location. On descent one wheel twisted upwards and almost tipped over. This is observed in near $ts = 3,250$.



(a) Test trial 1

Figure 6.7: Forecasts results for three test trials. *Cont.*

6.6 Hazard Detection

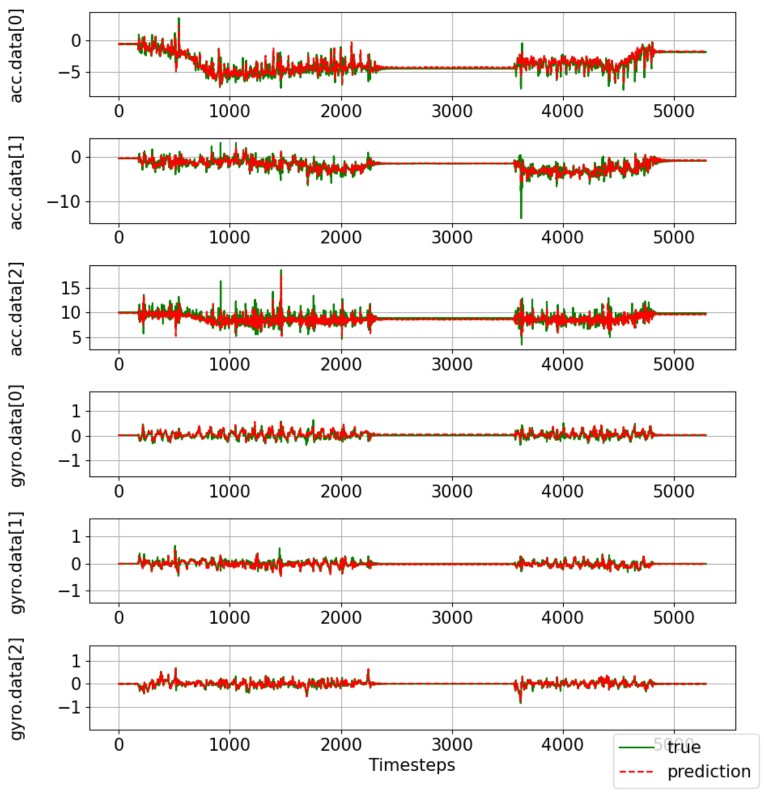
To evaluate hazard detection for the model the test loss is calculated for the predicted output. Fundamentally, anomalies are rare and is fortuitous if they can be observed in the data or in its distribution. However, it is usually assumed that these occur at the end of the distribution spectrum. Using the train data distribution, a set of thresholds are selected to establish if the predicted data corresponds to an anomaly. Figure 6.8 show these losses for each sensor reading.

Based on these distribution plots, a set of thresholds are defined for each sensor for each loss type. Table 6.4 shows these selections. The prediction loss is calculated for MSE and MAE as shown in equations 6.2 and 6.3.

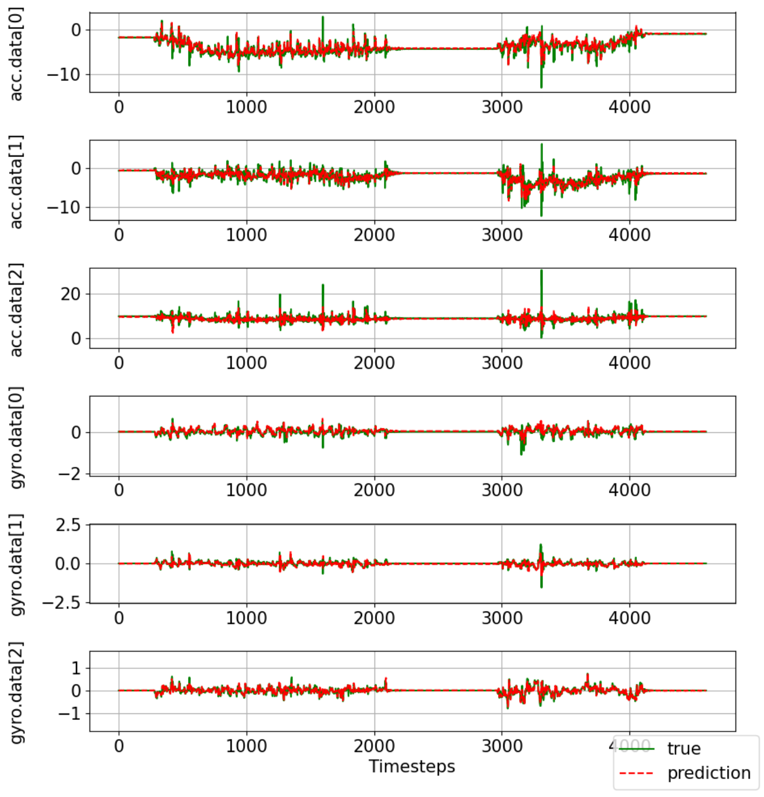
Loss	$accel^x$	$accel^y$	$accel^z$	$gyro^x$	$gyro^y$	$gyro^z$
MSE	0.04	0.04	0.04	0.02	0.05	0.04
MAE	0.25	0.25	0.25	0.25	0.25	0.25

Table 6.4: Selected thresholds for each loss type.

$$loss_{MSE} = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y})^2 \tag{6.2}$$

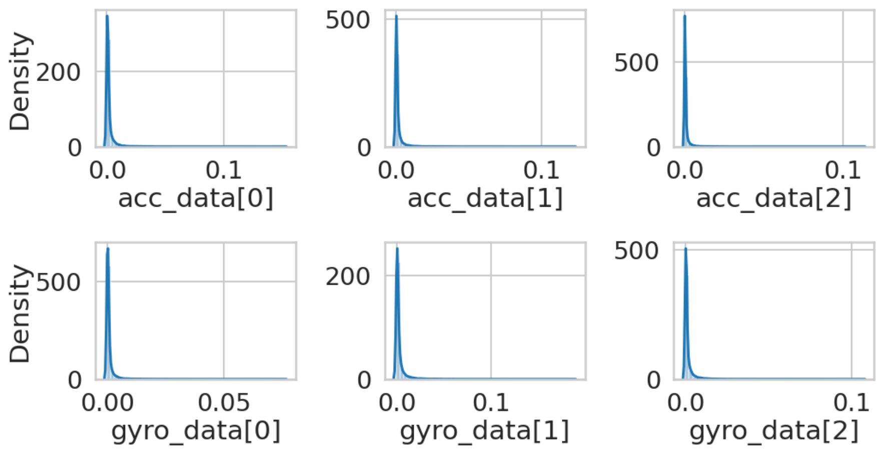


(b) Test trial 2

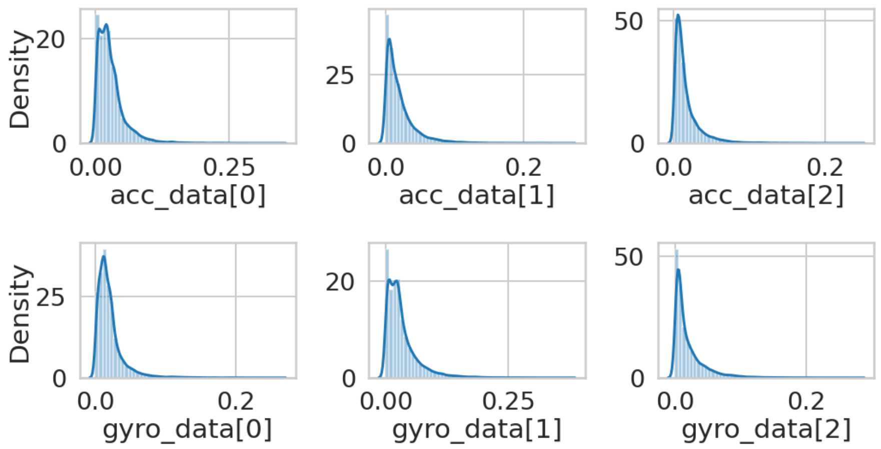


(c) Test trial 3

Figure 6.7: Forecasts results for three test trials.



(a) MSE loss distributions



(b) MAE loss distributions

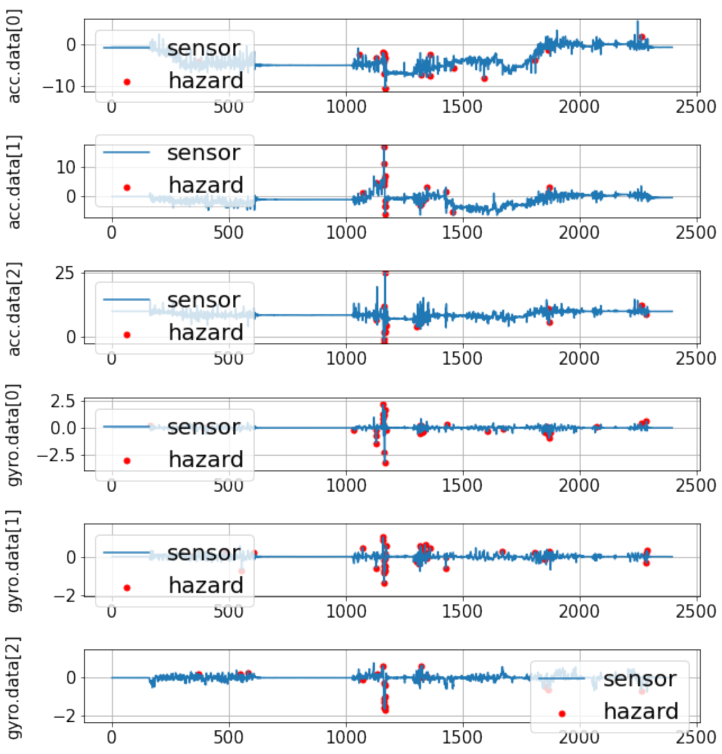
Figure 6.8: MSE loss distribution for the selected model.

$$loss_{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}|^2 \tag{6.3}$$

The anomaly detection is activated when the test loss passes the defined loss thresholds. The results obtained show interesting differences on the usage of the two losses. It is observed that each detect different nuances of anomalies. MSE penalizes errors more heavily than MAE. This is becomes influential to the solution of this problem for the following reasons. It is observed that the sensor readings for some axis overshoot when a motion that leads to the rover tipping over occurs and are therefore easily recognized by a human expert. For skidding or sliding motions however, this is a different story because these can be confused by the natural signatures the star-shaped wheels generate.

6.6.1 Detection with Mean Squared Error

Figures in 6.9 illustrate how the MSE-based detector is more sensitive to abrupt mobility changes in the AsguardIV rover. As previously described, the rover ascends and descends a slope. While it does this, the detector notices some minor issues while traversing the slopes furnished with obstacles and craters. In addition, it detects the important hazardous motions of interest in each of the test trials as well as the skid present in Test Trial 2. It is also noted that the the most abrupt behaviours are detected in the same time stamps across all sensor axis. This is most notable in Test trial 1.



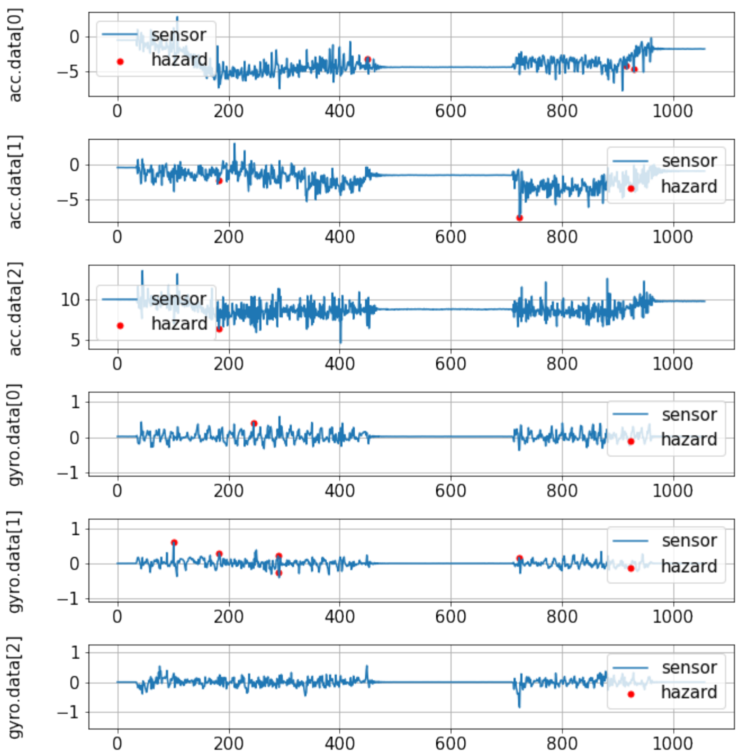
(a) Test trial 1

Figure 6.9: Hazard detection using MSE threshold values. *Cont.*

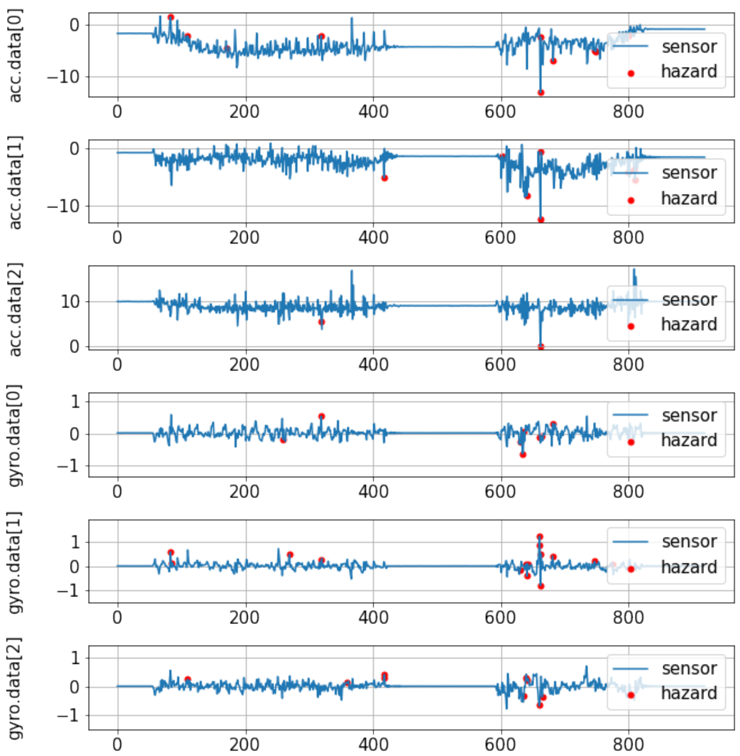
6.6.2 Detection with Mean Absolute Error

In contrast, Figure 6.10 illustrates how the MAE-based detector detects less minor abrupt motions but accurately detects hazardous movements leading to tip-over including the skidding event in Test trial no.2.

It is observed that detection is almost always registered consistently in the same time steps on all sensor axis. What this establishes is that the network has learned to detect these nuances for each sensor feature where it is relevant, i.e. for each feature. The comparison on the usage of MSE and MAE also demonstrate that both are useful depending on to the degree of hazard one would consider more important to characterize.

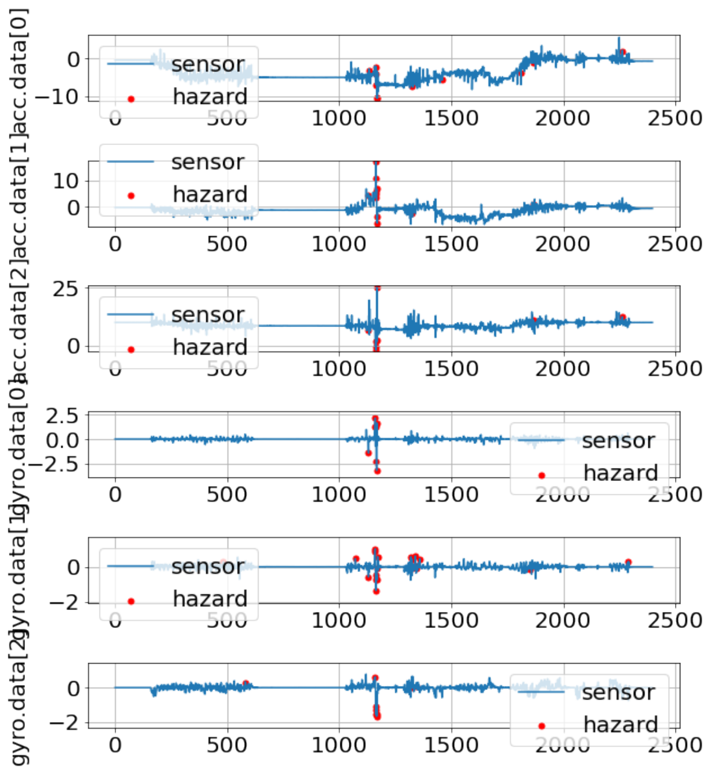


(b) Test trial 2

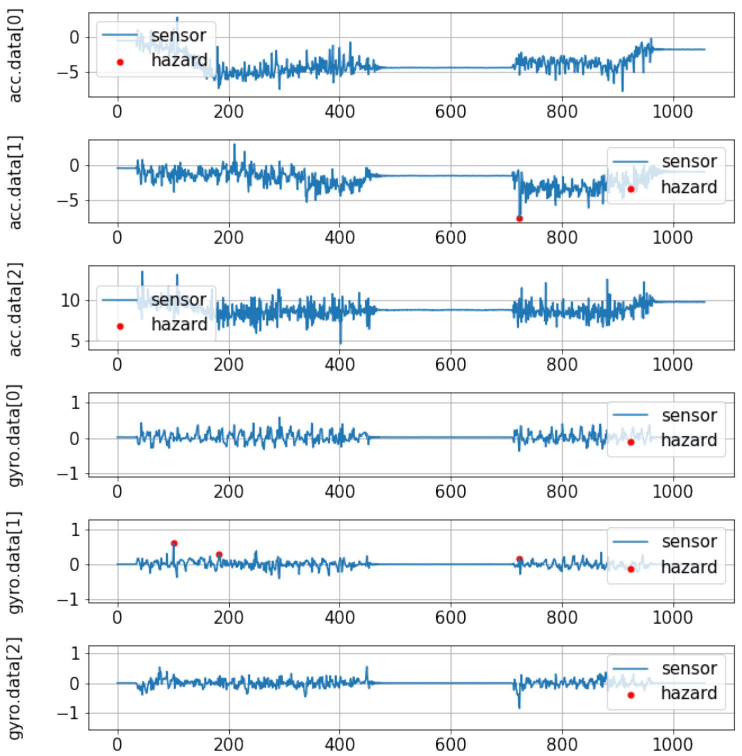


(c) Test trial 3

Figure 6.9: Hazard detection using MSE threshold values for three test trials.

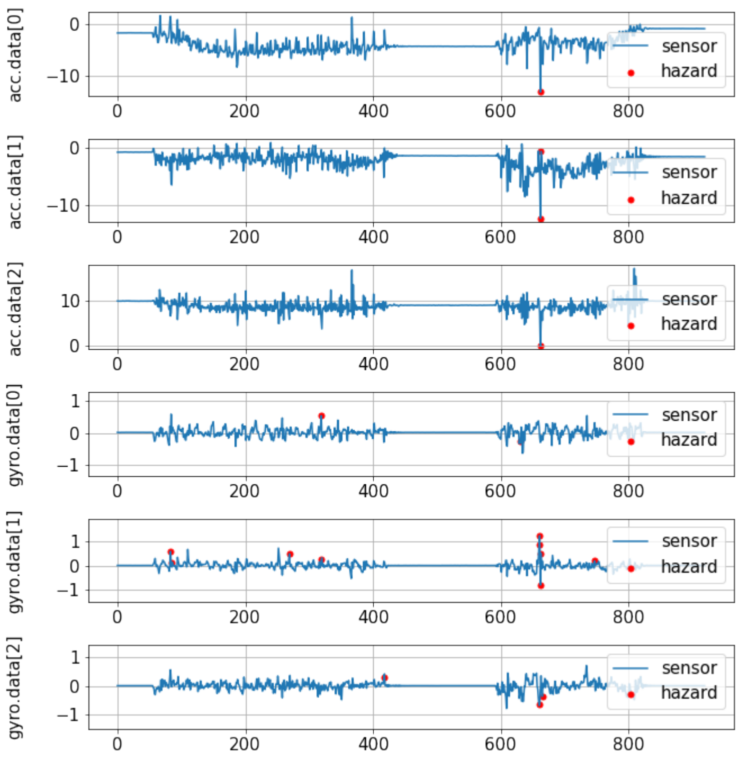


(a) Test trial 1



(b) Test trial 2

Figure 6.10: Hazard detection using MAE threshold values. *Cont.*



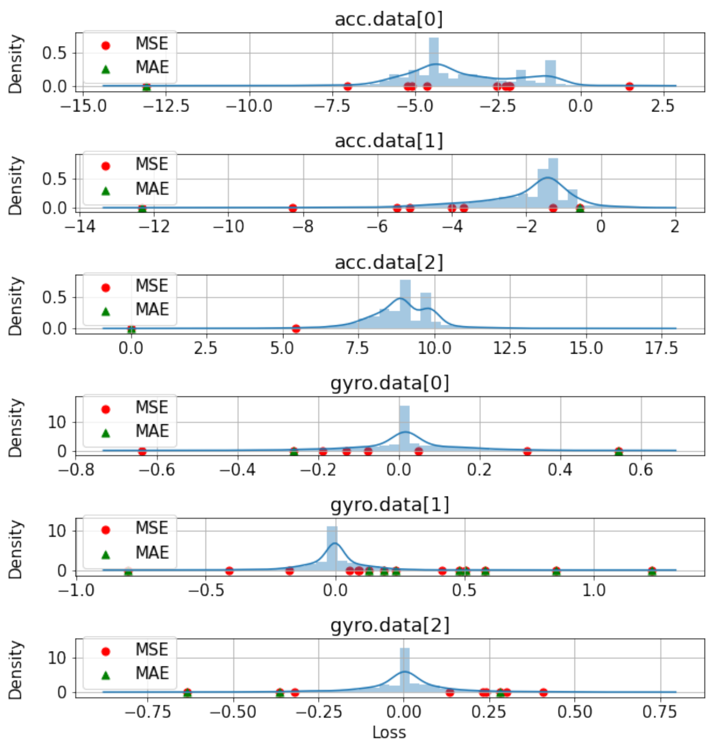
(c) Test trial 3

Figure 6.10: Hazard detection using MAE threshold values.

The performance of the detector needs to be subsumed under the evaluation of the forecast model. Some evaluation of anomaly detection methods suggest that in the absence of labels, these should be manufactured by the data user. In the case of this dataset, it is not required since it is visually possible to determine where the sensor jumps are located. These correspond to the sharp tip-over motions of the rover. However, in the case of the small bumping motions or skidding is not possible to determine in this dataset as they are not visually recognizable.

The author in Goix 2016 proposes an abnormality scoring function with which to evaluate the quality of unsupervised anomaly detection algorithms. While it has shown to often agree with known methods that require labels such as Receiver Operating Characteristic (ROC) and Precision-Recall (PR), the underlying assumption is that the data used corresponds to the normal or non-anomaly class. Naturally, this does not apply for this particular dataset. Doing so would require changing the framing of the problem altogether.

However, it is possible to perform a visual evaluation by creating histograms of the distribution of the sensor values and the detections to observe where they are located in the distribution. Plots shown in Figure 6.11 illustrates when in the distribution the detections are located. The distribution and histogram plots, like in all the plots before, show the values for each sensor axis separately. It is evident how the MAE-based thresholds detect hazards that fall generally in the tails of the distribution more than the MSE-based detector.



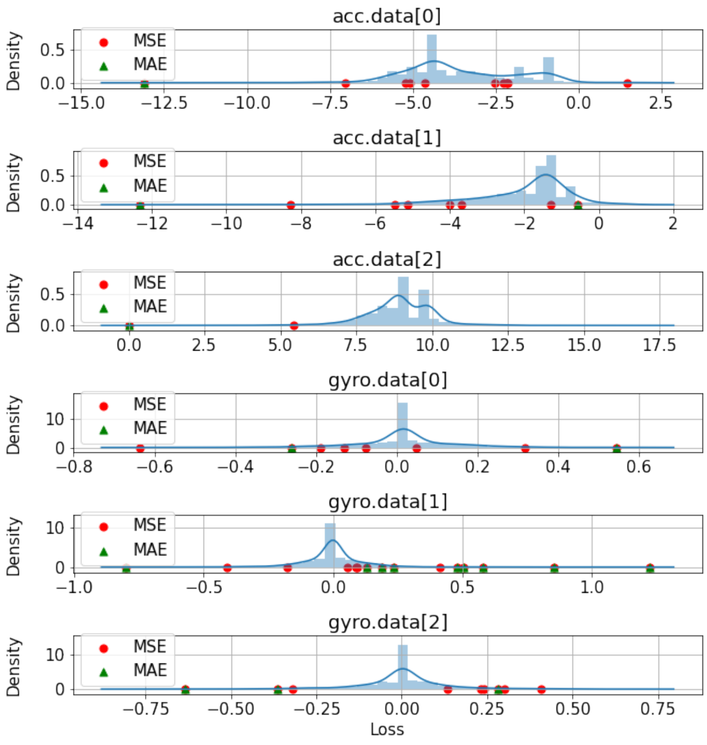
(a) Test trial 1

Figure 6.11: Hazard Detection Visual Evaluation. The figure illustrates the location in the distribution of the hazards detected by the MSE and MAE detectors, red and green respectively as shown in the legend. *Cont.*

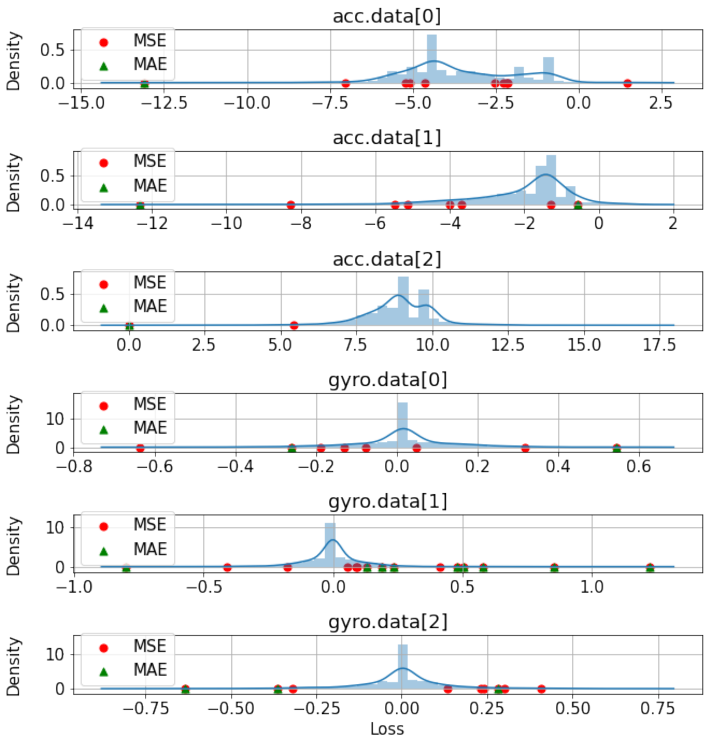
The latter detects hazards that are not limited to the tail of the distribution. It is important to keep in mind, however, that the data is composed of examples of all classes.

This shows that MSE picks up minor hazards related to the ascent and descent obstacles in the lunar analog and that the MAE does indeed picks up the less present more important hazards such as imminent tip-over. This demonstrates that the model itself is robust both in forecast and detection and reiterates the initial statement about the choice of loss in the detector depending on what types of hazards are of interest.

At the moment of deployment, the usage of equations 6.3 and 6.2 require an adjustment of the input data due to the argument that information about the future, i.e time step t, is not available. At the moment of evaluation on the test set, the loss equations require past information to yield a forecast sequence, but the loss equations require the current time step t to detect a hazard. This would simply not be possible. In practice, a 10ms would be required in advance to be able to forecast a mobility hazard. So, effectively the forecast would be performed with a 1 time step delay.



(b) Test trial 2



(c) Test trial 3

Figure 6.11: Hazard Detection Visual Evaluation. The figure illustrates the location in the distribution of the hazards detected by the MSE and MAE detectors, red and green respectively as shown in the legend.

6.7 Summary

This chapter presented a solution for hazardous motion forecast and detection for an AGV, the DFKIs AsguardIV rover. The motivation of this works stems from the need of enhanced autonomy modules for space rovers to overcome the challenges of distanced monitoring. For this unsupervised learning application, the goal is to forecast mobility behaviours and detect imminent hazards, e.g. instances in where the vehicle can potentially tip-over. The proposed solution is based on the combination of a LSTM-based Encoder-Decoder framework. The networks learn the traversing motions while traversing a reproduction of a lunar slope furnished with obstacles and craters with the objective of anticipating hazardous events.

The collected data corresponds to traverses on a moon crater analog environment where multiple challenges that match the real environment are present, e.g. slopes with different inclinations, rocks and boulders. In the evaluated data samples, the traverses cover surfaces with inclinations between $15°$ and $35°$. Hence, the scope of these experiments are bounded to mobility actions of ascent and descent of a rocky slope. This data was split into training and test sets.

A crucial challenge for this application was that the data is unlabeled and multiple motion behaviours are present in the samples. While some of which can be identifiable, minor mobilization hazards are not and therefore the data was kept whole and undivided. The proposed solution reaches a robust r^2 score of 0.969 under a one layer Encoder-Decoder LSTM framework of $5,060$ total parameters. In this manner, accomplishing the set goals of depth and robustness. While in the previous Chapter an important process for model selection was the use of SHAP values, this experiment had parameters independent of the BO sampling and it was easier to observe the effects of hyper-parametrization options by categorizing the results using the independent parameters. Thus, facilitating the model selection.

This application is an example on how hazard forecast and detection would be an invaluable attribute in an AI-enabled autonomy in space robotics for autonomous navigation scenarios and delivers a valuable contribution to the field of risk assessment for space rovers. This work has been submitted to the journal of Field Robotics as De Lucas Alvarez and Kirchner 2021 and is currently under review.

Chapter 7

Conclusion

"I am putting myself to the fullest possible use, which is all I think that any conscious entity can ever hope to do."
-HAL 9000, *2001: A Space Odyssey*

The autonomy of a robot depends greatly on the embedded routines that enable it to sense its status and its environment. Such ability has been explored from various perspectives in the scientific community. With the recent advancements in DL, many relevant problems in AI have been addressed with NNs and stand now as state-of-the-art solutions that have surpassed traditional ML methodologies.

The power consumption of such methods make their embedding into an autonomous vehicle prohibitive. However, the advent of processing units designed to equip mobile units, such as wearable devices and mobile robots, set a promising outlook for NN-enabled autonomous robots in the nearing future. This work is motivated by this observation and has focused on known and common situations encountered by autonomous robots during field operations.

7.1 Summary of Contributions

The main objective of this work was to develop solutions to known problems that are encountered during field operations by autonomous robots. Supported by standard analytical ML practices, these solutions are based on LSTMs, a type of NN specific for handling time series and sequences. A primary goal was to satisfy robustness and depth in the frameworks to support the opportunity of these solutions to be embedded in the vehicles. As the technology in processing units advances, it is undeniable that in the near future autonomous robots will supported by NN-enabled AI that optimizes power consumption without compromising accuracy.

The contributions of this work are high-performing NNs for important problems in field robotics achieving a number of goals, specifically:

- **High-performing shallow architectures.** The design of shallow neural networks that use relevant input to reduce training and processing time without compromising robust classification of states or tasks. Network depth requirements was fulfilled in all the problems presented in this work. This goal was accomplished for the three main field operation problems which had different levels of complexity:

1. **Hardware Failure Classification.** In the topic of sensor damage type identification in AUV thrusters, Chapter 4 proposes a shallow LSTM-based NN that outperforms the current stat-of-the-art in network depth and performance. A performance comparison of different methods that identify known thruster hardware failures from an AUV-emulating platform. This work presented feature selection by GMM modeling to reduce the input to the network and isolation through a HMM. The performance of these standard ML methodologies were used as baselines. In addition, a comparison of feature input effect on the performance was also presented. While previous works implemented (Ranganathan et al. 2001) rule-based solutions with different component terminals to infer the source or cause of sensor malfunction, this solution was capable of identifying the type of fault with a single sensor output in a shallow NN.
2. **Task Status Classification.** In the topic of trajectory classification for autonomous robots, Chapter 5 presents a performance comparison on different LSTM-based networks that classify real navigational trajectories in a AUV performing survey missions. In addition, the performance of the methods was evaluated based on sequence length, different balancing methods and parameter tuning. The latter involved the use of a method commonly used to explain learning of models, SHAP Values, but in this case, it was harnessed to observed the effect of network depth and size to choose the best model. This provided a more educated introspection into the selection of the networks based on its parameter values. This establishes a new state-of-the art for this lightly explored topic, where there is a need for shallow NN architectures that can be embedded in an autonomous vehicle for mission monitoring.
3. **Mobility Hazard Forecast and Detection.** In the field risk assessment for space rovers, Chapter 6 presents a performance comparison of different sizes and depths of LSTM Encoder-Decoder networks to forecast mobility actions of a space rover. The contribution to the state-of the-art comes in a model that does not require exteroceptive information to provide a hazard or risk prediction as a supervised learning problem (Skonieczny et al. 2019). The proposed LSTM Encoder uses only proprioceptive information from an IMU as an unsupervised learning problem. The NN is capable of anticipating a hazardous motion in time for possible prevention using up to 350ms of input data counting a lag input of 100ms. This would ensure that any programmed mobility action that threatens

the integrity of the mission or vehicle can be stopped in time. The detection component involves the performance analysis of two detectors based on MAE and MSE losses showed how they are successful at detecting different levels of hazards successfully.

- **Parameter importance for model selection.** The use of SHAP values is generally to explain a models feature learning. In this work, it is used in a different context to assist in the selection of models based on the tuned parameters. This methodology was developed in the conditions where hyper-parameter search was comprehensive and the execution of experiments relevant to class imbalance support the hyper-parameter search, as shown in Chapter 5. The adoption of this method assisted in selecting suitable models candidates for deployment supported by explainability methods.

7.2 Discussion and Learned Lessons

For the most part, the work presented in this dissertation demonstrates the viability of shallow NNs to support AI-enhanced autonomy in Autonomous Robots. As the three applications progressively increase in problem and methodology complexity some important points were identified to take into account in future work.

The constitution of the data can define the framing of the problem. The data specific to each problem also was influential in the way the research questions were approached. The compilation of the data is not always gathered by oneself and thus, depending on the nature of th problem

For instance, in the hardware failure application (Chapter 4), dealt with discrete outputs from a printed circuit board, where in a real thruster system output the signals could be continuous, while this may not always be the case for all systems. Despite this difference, it was still possible to model the data into a distribution to fit into the baseline ML methods. The manner of data collection also facilitated the classification as the RECOVERY dataset was designed so that each sample contained only one class.

For the task classification (Chapter 5) this was not the case, as the REGIME dataset was a collection of real missions which involved different AUV performing different trajectories. The nature of dataset made it possible to visually asses the data and segment it into classes, making supervised learning possible. In the third application (Chapter 6, the AsguardIV dataset increased the complexity of the problem. Even though this was also a collection of missions, each sample contains multiple known and unknown classes which are difficult to visually categorize. For this reason, unsupervised learning may still require some labeling for performance evaluation. Alternatively if the dataset allows, the samples could be used to train class specific models and build an ensemble solution.

Model tuning is not an art, but a methodical selective process. The hyper-parametrization tuning process progressed throughout the three applications from exhaustive to more optimized parameter selection methods. This choice increasingly accelerated training time and selection of good candidate networks. In addition, the combination of parameter tuning with resource allocation is also the best option for expediting the training, e.g. the use of HBBO (Chapter 6).

However not all tuning methods work the same for all frameworks. Regularization methods should be consistently in DL, the most common of such being Dropout. In the work presented in this dissertation Dropout showed to consistently worsen the performance of the LSTM networks. In the future it would be best to refrain from using Dropout with a shallow LSTMs as the network capacity is already small and therefore it can deteriorate performance. Other forms of regularization might be better suited when implementing LSTMs or CNNs, such as batch normalization.

Determining depth. While there are varied opinions regarding the performance of shallow vs. deep networks (Ba and Caruana 2013,Pascanu et al. 2014), the goal of this work was to demonstrate robust, shallow designs for these specific problems, where data is limited. As per the definition by Bengio 2009 the depth of a network refers to the number of levels of non-linear operations it performs. Shallow architectures are those which integrate 1, 2, or 3 levels. By this definition, the networks presented in this work comply with this characterization.

The design and implementation of these networks was not solely constrained to the depth but to rigorous and methodical tuning in order to minimize the number of parameters for each application without compromising performance metrics and in the future become potential energy efficient solutions for embedded ML-based AI in autonomous vehicles.

The design and optimization process of the networks presented here started off by assuming that the latter would already be shallow by limiting the search scope to such depths and sizes. Recent advances in Application Specific Integrated Circuits (ASICs) and Tensor Processing Units (TPUs) (Jouppi et al. 2017) have opened a research into that while highly relevant to this work lies a bit outside of scope as there were no such processing units embedded available in the mentioned autonomous robots. It does, however, pose relevant research questions for future work.

7.3 Future Work

This work can be further evaluated by implementing these solutions as part of the software suite embedded in an autonomous robot. Some recent works support the shift to data-centric solutions, where the ML algorithm runs where the data is originated, where NNs can be implemented in small integrated circuits, (Ando et al. 2017; Bankman et al. 2018). While recent work on the combination of dynamic control information to enrich the information supplied to the NN (Wehbe and Krell 2017; Wehbe, Arriaga, et al. 2018), it would require an optimization study to asses the viability of such solutions to be distilled and compressed into more shallow networks. There has been some work in this field of strategies for pruning and compressing are used for Deep Neural Networks (DNNs) (Cheng, Wang, et al. 2020; Liu et al. 2020).

It is also of interest to further study the integration of forecast, classification and detection into a single framework. As shown in Chapter 6, the forecast model output was used to synthesize a straight-forward hazard detector. In this manner, the depth of the network is not compromised. A comprehensive solution such as this would certainly need to also be optimized to preserve the desired depth.

Being these meant to be data-centric solutions, the integration of continual learning for their performance enhancement would also benefit the work in this dissertation. Recent work in developmental systems based in NNs (Parisi et al. 2019) show the relevance for RAS to update over time their knowledge as they interact with their environment. For example, such methods could benefit from learning unknown anomalous or non-anomalous navigational trajectories (Chapters 4, 5) or unexpected mobility hazards (Chapter 6), as not all possible events can be accounted for.

Bibliography

Alessandri, A., M. Caccia, and G. Veruggio (1998). “Model-based approach to fault diagnosis in unmanned underwater vehicles”. In: *Oceans Conference Record (IEEE)*. Vol. 2. IEEE, pp. 825–827.

Alessandri, A., M. Caccia, and G. Veruggio (1999). “Fault detection of actuator faults in unmanned underwater vehicles”. In: *Control Engineering Practice* 7.3, pp. 357–368.

Ando, Kota et al. (2017). “BRein memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm CMOS”. In: *IEEE Symposium on VLSI Circuits, Digest of Technical Papers*. Institute of Electrical and Electronics Engineers Inc., pp. C24–C25.

Ba, Lei Jimmy and Rich Caruana (2013). “Do Deep Nets Really Need to be Deep?” In: *Advances in Neural Information Processing Systems* 3.January, pp. 2654–2662.

Banerjee, S., J. Harrison, P. M. Furlong, and M. Pavone (2020). *Adaptive Meta-Learning for Identification of Rover-Terrain Dynamics*. Tech. rep.

Bankman, Daniel, Lita Yang, Bert Moons, Marian Verhelst, and Boris Murmann (2018). “An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS”. In: *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*. Vol. 61. Institute of Electrical and Electronics Engineers Inc., pp. 222–224.

Bashir, Faisal I. and Ashfaq A. Khokhar (2007). “Object Trajectory-Based Activity Classification and Recognition using Hidden Markov Models”. In: *IEEE Transactions on Image Processing*, pp. 1912–1919.

Beer, Jenay M, Arthur D Fisk, and Wendy A Rogers (2014). “Toward a framework for levels of robot autonomy in human-robot interaction.” In: *Journal of human-robot interaction* 3.2, pp. 74–99.

Bengio, Y., P. Frasconi, and P. Simard (1993). “The problem of learning long-term dependencies in recurrent networks”. In: *IEEE International Conference on Neural Networks*. IEEE, pp. 1183–1188.

Bengio, Yoshua (2009). *Learning Deep Architectures for AI*. now Publishers Inc.

Bengio, Yoshua, Paolo Frasconi, and Patrice Simard (1994). "Learning Long-Term Depencencies with Gradien Descent is Difficult". In: *IEE International Conferece on Neural Networks*. San Francisco: IEEE Press, pp. 1183–1195.

Bergstra, J., D. Yamins, and D. Cox (2013). *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures*. Tech. rep.

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (1994). *Algorithms for Hyper-Parameter Optimization*. Tech. rep.

Bergstra, James and Yoshua Bengio (2012). "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research*. Vol. 13, pp. 281–305.

Bian, Xinqian, Tao Chen, Zheping Yan, Dehui Zhao, and Guang Yu (2009). "Fault diagnosis based on grey dynamic prediction for AUV sensor". In: *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 1–6.

Biljecki, Filip (2012). *Automatic segmentation and classification of movement trajectories for transportation modes*.

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc.

Bono, R., Ga Bruzzone, Gi Bruzzone, and M. Caccia (1999). "ROV actuator fault diagnosis through servo-amplifiers' monitoring: an operational experience". In: *Oceans Conference Record (IEEE)*. Vol. 3. IEEE, pp. 1318–1324.

Bouguelia, Mohamed Rafik, Ramon Gonzalez, Karl Iagnemma, and Stefan Byttner (2017). "Unsupervised classification of slip events for planetary exploration rovers". In: *Journal of Terramechanics* 73, pp. 95–106.

Bozzano, M, A Cimatti, M Roveri, A Tchaltsev Fondazione, and Bruno Kessler (2011). "A Comprehensive Approach to On-Board Autonomy Verification and Validation". In: *International Joint Conference on Artificial Intelligence A*. IJCAI'11. AAAI Press, pp. 2398–2403.

Byon, Young-Ji, Baher Abdulhai, and Amer Shalaby (2009). "Real-Time Transportation Mode Detection via Tracking Global Positioning System Mobile Devices". In: *Journal of Intelligent Transportation Systems* 13.4, pp. 161–170.

Chaudhari, Pratik et al. (2016). "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys". In: *CoRR*.

Chen, Xiaolong, Yuru Xu, Lei Wan, and Ye Li (2010). "Sensor fault diagnosis for autonomous underwater vehicle". In: *Proceedings - 2010 7th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010*. Vol. 6, pp. 2918–2923.

Cheng, Min, Qian Xu, et al. (2016). "MS-LSTM: A multi-scale LSTM model for BGP anomaly detection". In: *Proceedings - International Conference on Network Protocols, ICNP*. NetworkML, pp. 1–6.

Cheng, Yu, Duo Wang, Pan Zhou, and Tao Zhang (2020). *A Survey of Model Compression and Acceleration for Deep Neural Networks*. Tech. rep.

Cho, Kyunghyun et al. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. Tech. rep.

Corradini, M. L., A. Monteriu, G. Orlando, and S. Pettinari (2011). "An actuator failure tolerant robust control approach for an underwater Remotely Operated Vehicle". In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 3934–3939.

De Lucas Alvarez, Mariela, Helen Hastie, and David Lane (2017). "Navigation-Based learning for survey trajectory classification in autonomous underwater vehicles". In: *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, pp. 1–6.

De Lucas Alvarez, Mariela and Frank Kirchner (2021). "Hazardous Mobility Forecast and Detection in Space Rovers". Unpublished.

De Lucas Alvarez, Mariela and David Lane (2016). "A Hidden Markov Model application with Gaussian Mixture emissions for fault detection and diagnosis on a simulated AUV platform". In: *OCEANS 2016 MTS/IEEE Monterey*, pp. 1–4.

Dearden, Richard and Juhan Ernits (2013). "Automated fault diagnosis for an autonomous underwater vehicle". In: *IEEE Journal of Oceanic Engineering*. Vol. 38. 3, pp. 484–499.

Dimastrogiovanni, Mauro, Florian Cordes, and Giulio Reina (2020). "Terrain estimation for planetary exploration robots". In: *Applied Sciences (Switzerland)* 10.17, p. 6044.

Dodge, Somayeh, Robert Weibel, and Ehsan Forootan (2009). "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects". In: *Computers, Environment and Urban Systems* 33.6, pp. 419–434.

Domínguez, Raúl, Sascha Arnold, Christoph Hertzberg, and Arne Böckmann (2018). "Internal Simulation for Autonomous Robot Exploration of Lava Tubes". In: *In Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 1:, Porto, Portugal*, pp. 144–155.

Endsley, Mica R. and David B. Kaber (1999). "Level of automation effects on performance, situation awareness and workload in a dynamic control task". In: *Ergonomics* 42.3, pp. 462–492.

Erfani, Sarah M, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie (2016). "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning". In: *Pattern Recognition*. Vol. 58, pp. 121–134.

Fagogenis, Georgios, Valerio De Carolis, and David M. Lane (2016). "Online fault detection and model adaptation for Underwater Vehicles in the case of thruster failures". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2625–2630.

Falkner, Stefan, Aaron Klein, and Frank Hutter (2018). "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". In: *35th International Conference on Machine Learning, ICML 2018*. Vol. 4, pp. 2323–2341.

Fang, Yixin (2011). "Asymptotic Equivalence between Cross-Validations and Akaike Information Criteria in Mixed-Effects Models". In: *Journal of Data Science* 9, pp. 15–21.

Ferrell, Bob et al. (2010). "Usage of Fault Detection Isolation & Recovery (FDIR) in Constellation (CxP) Launch Operations". In: *SpaceOps 2010 Conference*, pp. 1–28.

García, Jesus, Oscar Pérez Concha, José M. Molina, and Gonzalo De Miguel (2006). "Trajectory classification based on machine-learning techniques over tracking data". In: *2006 9th International Conference on Information Fusion, FUSION*, pp. 1–8.

Gaura, E. and M. Kraft (2002). "Are neural network techniques the solution to measurement validation, monitoring and automatic diagnosis of sensor faults?" In: *41st SICE Annual Conference (SICE)*. Institute of Electrical and Electronics Engineers (IEEE), pp. 2052–2057.

Gers, F.A. and J. Schmidhuber (2000). "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. IEEE, 189–194 vol.3.

Giannotti, Fosca, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi (2007). "Trajectory pattern mining". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 330–339.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. Proceedings of Machine Learning Research, pp. 249–256.

Goix, Nicolas (2016). *How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms?* Tech. rep.

Gonzalez, Ramon, Dimi Apostolopoulos, and Karl Iagnemma (Mar. 2018). “Slippage and immobilization detection for planetary exploration rovers via machine learning and proprioceptive sensing”. In: *Journal of Field Robotics* 35.2, pp. 231–247.

Gonzalez, Ramon, Samuel Chandler, and Dimi Apostolopoulos (2019). “Characterization of machine learning algorithms for slippage estimation in planetary exploration rovers”. In: *Journal of Terramechanics* 82, pp. 23–34.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Goyal, Priya et al. (2017). “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. In: *CoRR*.

Graves, Alex (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. Studies in Computational Intelligence. Springer.

Greff, Klaus, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber (2017). “LSTM: A Search Space Odyssey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, pp. 2222–2232.

Hamilton, Kelvin, Dave Lane, Nick Taylor, and Keith Brown (2001). “Fault diagnosis on autonomous robotic vehicles with RECOVERY: An integrated heterogeneous-knowledge approach”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 4, pp. 3232–3237.

Hardt, Moritz and Tengyu Ma (2016). “Identity Matters in Deep Learning”. In: *CoRR*.

Hinton, G E, N Srivastava, A Krizhevsky, I Sutskever, and R R Salakhutdinov (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. Tech. rep.

Hinton, Geoffrey (1988). *Neural Network Architectures for Artificial Intelligence*. USA: American Association for Artificial Intelligence.

Hochreiter, Sepp (1991). *DIPLOMARBEIT IM FACH INFORMATIK Untersuchungen zu dynamischen neuronalen Netzen*. Tech. rep.

Hochreiter, Sepp and Jürgen Schmidhuber (1997a). “Flat Minima”. In: *Neural Computation* 9.1, pp. 1–42.

Hochreiter, Sepp and Jürgen Schmidhuber (1997b). “Long Short-Term Memory”. In: *Neural Comput.* 9.8, pp. 1735–1780.

Hoffer, Elad, Itay Hubara, and Daniel Soudry (2017). “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”. In: *Advances in Neural Information Processing Systems* 30, pages 1729–1739.

Hsu, Daniel (2017). *Anomaly Detection on Graph Time Series*. Tech. rep.

Hussain, Saed, Maizura Mokhtar, and Joe M. Howe (2015). "Sensor failure detection, identification, and accommodation using fully connected cascade neural network". In: *IEEE Transactions on Industrial Electronics*. Vol. 62. 3. Institute of Electrical and Electronics Engineers Inc., pp. 1683–1692.

Innocenti, M and Marcello Napolitano (2002). "Neural Networks and other Techniques for Fault Identification and Isolation of Aircraft Systems". In: *Aerospace Engineering*. May, pp. 13–17.

Inotsume, Hiroaki, Masataku Sutoh, Kenji Nagaoka, Keiji Nagatani, and Kazuya Yoshida (2013). "Modeling, Analysis, and Control of an Actively Reconfigurable Planetary Rover for Traversing Slopes Covered with Loose Soil". In: *Journal of Field Robotics* 30.6, pp. 875–896.

Jack, L. B. and A. K. Nandi (2002). "Fault detection using support vector machines and artificial neural networks, augmented by genetic algorithms". In: *Mechanical Systems and Signal Processing* 16.2-3, pp. 373–390.

Jager, Georg et al. (2014). "Assessing neural networks for sensor fault detection". In: *CIVEMSA 2014 - 2014 IEEE Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications, Proceedings*. IEEE Computer Society, pp. 70–75.

Jamieson, Kevin and Ameet Talwalkar (2016). *Non-stochastic Best Arm Identification and Hyperparameter Optimization*. Tech. rep.

Jia, Qilong, Jinxue Xu, and Guofeng Wang (2013a). "Fault diagnosis based on grey correlation analysis for autonomous underwater vehicle sensor". In: *Proceedings - 2013 Chinese Automation Congress, CAC 2013*. IEEE Computer Society, pp. 656–659.

Jia, Qilong, Jinxue Xu, and Guofeng Wang (2013b). "Fault diagnosis based on second-order Taylor series dynamic prediction for autonomous underwater vehicle sensor". In: *Proceedings - 2013 Chinese Automation Congress, CAC 2013*. IEEE Computer Society, pp. 651–655.

Jomeiri, Alireza (2010). "Software fault detection for reliability using recurrent neural network modeling". In: *ICSTE 2010 - 2010 2nd International Conference on Software Technology and Engineering, Proceedings*. Vol. 2, pp. 149–152.

Jones, Donald R (2001). "A Taxonomy of Global Optimization Methods Based on Response Surfaces". In: *Journal of Global Optimization*. Vol. 21, pp. 345–383.

Jouppi, Norman P et al. (2017). "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. Toronto, ON, Canada: Association for Computing Machinery, pp. 1–12.

Jozefowicz, Rafal and Wojciech Zaremba (2015). *An Empirical Exploration of Recurrent Network Architectures*. Tech. rep.

Juba, Brendan, Christopher Musco, Fan Long, Stelios Sidiroglou-Douskos, and Martin Rinard (2015). "Long Short Term Memory Networks for Anomaly Detection in Time Series". In: *Proceedings 2015 Network and Distributed System Security Symposium*. April, pp. 22–24.

Kawatsuma, Shinji, Mineo Fukushima, and Takashi Okada (2012). "Emergency response by robots to Fukushima-Daiichi accident: summary and lessons learned". In: *Industrial Robot: An International Journal* 39.5, pp. 428–435.

King, Gary et al. (2001). *Logistic Regression in Rare Events Data*. Tech. rep.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, pp. 1–15.

Kiranyaz, Serkan et al. (2019). "1D Convolutional Neural Networks and Applications: A Survey". In: *Mechanical Systems and Signal Processing*. Vol. 151, pp. 1–21.

Köhler, Tim, Elmar Berghöfer, Christian Rauch, and Frank Kirchner (2013). "Sensor Fault Detection and Compensation in Lunar/Planetary Robot Missions Using Time-Series Prediction Based on Machine Learning". In: *Acta Futura: AI in Space Workshop at IJCAI*. Vol. 9, pp. 9–20.

Kruger, Justin, Arno Rogg, and Ramon Gonzalez (2019). "Estimating Wheel Slip of a Planetary Exploration Rover via Unsupervised Machine Learning". In: *IEEE Aerospace Conference Proceedings*. Vol. 2019-March. IEEE Computer Society, pp. 1–8.

Lane, David M. et al. (2012). "Persistent autonomy: The challenges of the PANDORA project". In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 9. PART 1. IFAC Secretariat, pp. 268–273.

Lee, Jae-Gil, Jiawei Han, Xiaolei Li, and Hector Gonzalez (2008). "TraClass: Trajectory Classification Using Hierarchical Region-Based and Trajectory-Based Clustering". In: *Proceedings of the VLDB Endowment*. Vol. 1. 1. VLDB Endowment, pp. 1081–1094.

Lee, Jae-Gil, Jiawei Han, and Kyu-Young Whang (2007). *Trajectory Clustering: A Partition-and-Group Framework*. SIGMOD '07. Association for Computing Machinery, pp. 593–604.

Lee, Jangwon and Michael S. Ryoo (2017). "Learning Robot Activities from First-Person Human Videos Using Convolutional Future Regression". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 472–473.

Li, Lisha, Kevin Jamieson, Afshin Rostamizadeh, and Ameet Talwalkar (2018). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research*. Vol. 18, pp. 1–52.

Li, Xun (2014). "Using Complexity Measures of Movement for Automatically Detecting Movement Types of Unknown GPS Trajectories". In: *American Journal of Geographic Information System* 2.3, pp. 63–74.

Liao, Lin, Dieter Fox, and Henry Kautz (2007). *Learning and Inferring Transportation Routines*. Tech. rep.

Liao, Lin, Donald J Patterson, Dieter Fox, and Henry Kautz (2006). *Building Personal Maps from GPS Data*. Tech. rep.

Lipton, Zachary C., David C. Kale, Charles Elkan, and Randall Wetzel (2016). "Learning to diagnose with LSTM recurrent neural networks". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, pp. 1–18.

Liu, Jiayi, Samarth Tripathi, Unmesh Kurup, and Mohak Shah (2020). *Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey*. Tech. rep.

Lu, Weining et al. (2017). "Unsupervised Sequential Outlier Detection with Deep Architectures". In: *IEEE Transactions on Image Processing* 26.9, pp. 4321–4330.

Lundberg, Scott M and Paul G Allen (2018). *Consistent feature attribution for tree ensembles*. Tech. rep.

Lundberg, Scott M, Paul G Allen, and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *31st Conference on Neural Information Processing Systems*. Long Beach, CA, USA, pp. 4768–4777.

Medel, Jefferson Ryan and Andreas Savakis (2016). *Anomaly Detection Using Predictive Convolutional Long Short-Term Memory Units*. Tech. rep.

Mlích, Jozef and Petr Chmelar (2008). "Trajectory classification based on hidden markov models". In: *Proceedings of 18th International Conference on Computer Graphics and Vision*, pp. 101–105.

Molnar, Christoph (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*.

Monteriù, Andrea, Prateek Asthan, Kimon Valavanis, and Sauro Longhi (2007a). "Model-based sensor fault detection and isolation system for unmanned ground vehicles: Experimental validation (part II)". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2744–2751.

Monteriù, Andrea, Prateek Asthan, Kimon Valavanis, and Sauro Longhi (2007b). "Model-based sensor fault detection and isolation system for unmanned ground vehicles: Theoretical aspects (part I)". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2736–2743.

Nascimento, Jacinto C., Mário A.T. Figueiredo, and Jorge S. Marques (2010). "Trajectory classification using switched dynamical hidden markov models". In: *IEEE Transactions on Image Processing* 19.5, pp. 1338–1348.

Obst, Oliver (2013). "Distributed Fault Detection in Sensor Networks using a Recurrent Neural Network". In: *Neural Processing Letters*. Vol. 40. 3. Kluwer Academic Publishers, pp. 261–273.

Olah, C. (2015). *Understanding LSTM Networks*.

Omerdic, Edin and Geoff Roberts (2004). "Thruster fault diagnosis and accommodation for open-frame underwater vehicles". In: *Control Engineering Practice* 12.12 SPEC. ISS. Pp. 1575–1598.

Osman, Asmaa A.E., Reda A. El-Khoribi, Mahmoud E. Shoman, and M. A. Wahby Shalaby (2017). "Trajectory learning using posterior hidden Markov model state distribution Posterior Hidden Markov Model State Distribution". In: *Egyptian Informatics Journal* 18.3, pp. 171–180.

Panagiotakis, Costas, Nikos Pelekis, and Ioannis Kopanakis (2009). "Trajectory voting and classification based on spatiotemporal similarity in moving object databases". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5772 LCNS. Springer, Berlin, Heidelberg, pp. 131–142.

Parisi, German I., Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter (2019). "Continual lifelong learning with neural networks: A review". In: *Neural Networks*. Vol. 113. Elsevier Ltd, pp. 54–71.

Pascanu, Razvan, Guido Montúfar, Mont´ Montúfar, and Yoshua Bengio (2014). *On the number of response regions of deep feedforward networks with piecewise linear activations*. Tech. rep.

Podder, Tarun Kanti and Nilanjan Sarkar (2001). "Fault-tolerant control of an autonomous underwater vehicle under thruster redundancy". In: *Robotics and Autonomous Systems* 34.1, pp. 39–52.

Qin, Z. and J. Gu (2009). "Sensor Fault Detection and Identification Based on Gray model for Autonomous Underwater Vehicle". In: *The Mediterranean Journal of Measurement and Control*. Vol. 5. 2, pp. 71–77.

Quinlan, J. Ross (1993). *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Raanan, Ben Yair et al. (2016). “Automatic fault diagnosis for autonomous underwater vehicles using online topic models”. In: *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*. Institute of Electrical and Electronics Engineers Inc., pp. 1–6.

Rabiner, Lawrence R. (1989). “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE*, pp. 257–286.

Rae, Graeme J.S. and Stan E. Dunn (1994). “On-line damage detection for autonomous underwater vehicles”. In: *IEEE Sympsium on Autonomous Underwater Vehicle Technology*. IEEE, pp. 383–392.

Ranganathan, N., Minesh I. Patel, and R. Sathyamurthy (2001). *An intelligent system for failure detection and control in an autonomous underwater vehicle*.

Reddy, Sasank, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava (2008). “Determining transportation mode on mobile phones”. In: *Proceedings - International Symposium on Wearable Computers, ISWC*, pp. 25–28.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). *”Why Should I Trust You?”: Explaining the Predictions of Any Classifier*. Tech. rep.

Robnik-Šikonja, Marko and Marko Bohanec (2018). “Perturbation-Based Explanations of Prediction Models”. In: *Human and Machine Learning: Visible, Explainable, Trustworthy and Transparent*. Springer International Publishing, pp. 159–175.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning representations by back-propagating errors”. In: *Nature* 323.6088, pp. 533–536.

Russell, Stuart J., Peter Norvig, and John. Canny (2014). *Artificial intelligence : a modern approach*, p. 1091.

SAE International (2014). *Summary of SAE International’s Levels of Driving Automation for On-Road Vehicles*. Tech. rep. SAE International, p. 12.

Sas, Corina, Gregory O’Hare, and Ronan Reilly (2003). “Online trajectory classification”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2659, pp. 1035–1044.

Schmidhuber, Jürgen (2015). “Deep Learning in Neural Networks: An overview”. In: *Neural Networks*. Vol. 61. Elsevier Ltd, pp. 85–117.

Selmic, Rastko R., Marios M. Polycarpou, and Thomas Parisini (2006). “Actuator fault detection in nonlinear uncertain systems using neural on-line approximation models”.

In: *Proceedings of the American Control Conference*. Vol. 2006. Institute of Electrical and Electronics Engineers Inc., pp. 5123–5128.

Shahriari, Bobak, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas (2016). *Taking the Human Out of the Loop: A Review of Bayesian Optimization*. Tech. rep.

Shapley, Lloyd S. (1953). “A value for n-person games”. In: *Contributions to the Theory of Games* 2.28, pp. 307–3.

Shirish Keskar, Nitish et al. (2016). *On large-batch training for deep learning: generalization gap and sharp minima*. Tech. rep.

Simani, S., F. Marangon, and C. Fantuzzi (1999). “Fault diagnosis in a power plant using artificial neural networks: Analysis and comparison”. In: *European Control Conference, ECC 1999 - Conference Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 2270–2275.

Skonieczny, Krzysztof, Dhara K. Shukla, Michele Faragalli, Matthew Cole, and Karl D. Iagnemma (2019). “Data-driven mobility risk prediction for planetary rovers”. In: *Journal of Field Robotics* 36.2, pp. 475–491.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, and Ruslan Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research*. Vol. 15, pp. 1929–1958.

Stanway, M. Jordan et al. (2015). “White shark strike on a long-range AUV in Monterey Bay”. In: *MTS/IEEE OCEANS 2015 - Genova: Discovering Sustainable Ocean Energy for a New World*. Institute of Electrical and Electronics Engineers Inc.

Stone, M. (1977). “An Asymptotic Equivalence of Choice of Model by Cross-Validation and Akaike’s Criterion”. In: *Journal of the Royal Statistical Society. Series B (Methodological)*. Vol. 39. WileyRoyal Statistical Society, pp. 44–47.

Štrumbelj, Erik and Igor Kononenko (2014). “Explaining prediction models and individual predictions with feature contributions”. In: *Knowl Inf Syst* 41, pp. 647–665.

Stubbs, Kristen, Pamela J. Hinds, and David Wettergreen (2007). “Autonomy and Common Ground in Human-Robot Interaction: A Field Study”. In: *IEEE Intelligent Systems* 22.2, pp. 42–50.

Sundararajan, Mukund and Amir Najmi (2020). “The Many Shapley Values for Model Explanation”. In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 9269–9278.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). *Sequence to Sequence Learning with Neural Networks*. Tech. rep.

Wang, Jianguo, Gongxing Wu, and Lei Wan (2008). "Sensor fault diagnosis for underwater robots". In: *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pp. 254–259.

Wang, Yujia, Jie Zhao, and Mingjun Zhang (2008). "Research on the Sensors Condition Monitoring Method for AUV". In: *Proceedings of the First International Conference on Intelligent Robotics and Applications: Part I*. Vol. 5314. ICIRA '08. Springer-Verlag, pp. 427–436.

Wehbe, Bilal, Octavio Arriaga, Mario Michael Krell, and Frank Kirchner (2018). "Learning of Multi-Context Models for Autonomous Underwater Vehicles". In: *AUV 2018 - 2018 IEEE/OES Autonomous Underwater Vehicle Workshop, Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 1–6.

Wehbe, Bilal, Marc Hildebrandt, and Frank Kirchner (2017). "Experimental evaluation of various machine learning regression methods for model identification of autonomous underwater vehicles". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 0. Institute of Electrical and Electronics Engineers Inc., pp. 4885–4890.

Wehbe, Bilal and Mario Michael Krell (2017). "Learning coupled dynamic models of underwater vehicles using Support Vector Regression". In: *OCEANS 2017 - Aberdeen*. Vol. 2017-Octob. Institute of Electrical and Electronics Engineers Inc., pp. 1–7.

Wong, Cuebong, Erfu Yang, Xiu Tian Yan, and Dongbing Gu (2017). "An overview of robotics and autonomous systems for harsh environments". In: *ICAC 2017 - 2017 23rd IEEE International Conference on Automation and Computing: Addressing Global Challenges through Automation and Computing*. Institute of Electrical and Electronics Engineers Inc., pp. 1–6.

Xu, Xiao and J Wesley Hines (1999). "Sensor Validation and Fault Detection Using Neural Networks". In: *Proc. Maintenance and Reliability Conference (MARCON 99)*, pp. 1–9.

Zhang, Mingjun, Juan Wu, and Yujia Wang (2009). "Sensor soft fault detection method of autonomous underwater vehicle". In: *2009 IEEE International Conference on Mechatronics and Automation, ICMA 2009*, pp. 4839–4844.

Zhang, Yu, Chris Bingham, Zhijing Yang, Michael Gallimore, and Paul Stewart (2012). "Applied Sensor Fault Detection and Identification Using Hierarchical Clustering and SOMNNs, with Faulted-Signal Reconstruction". In: *Proceedings of 15th International Conference MECHATRONIKA*, pp. 1–7.

Zheng, Yu, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma (2010). “Understanding transportation modes based on GPS data for Web applications”. In: *ACM Transactions on The Web*. Vol. 4. 1, pp. 247–256.

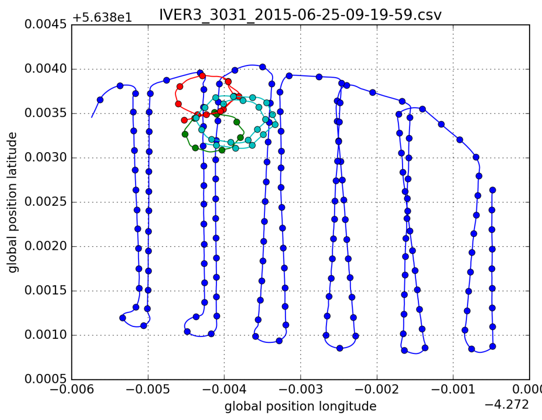
Zheng, Yu, Like Liu, Longhao Wang, and Xing Xie (2008). “Learning Transportation Mode from Raw GPS Data for Geographic Applications on the Web”. In: *Proceedings of the 17th International Conference on World Wide Web*. Beijing, China: Association for Computing Machinery, pp. 247–256.

Appendix A

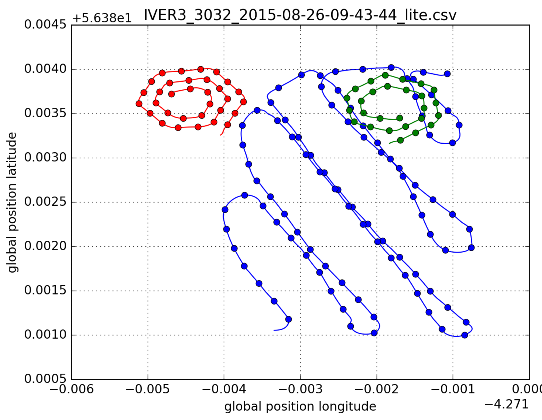
Figures

A.1 Chapter 5

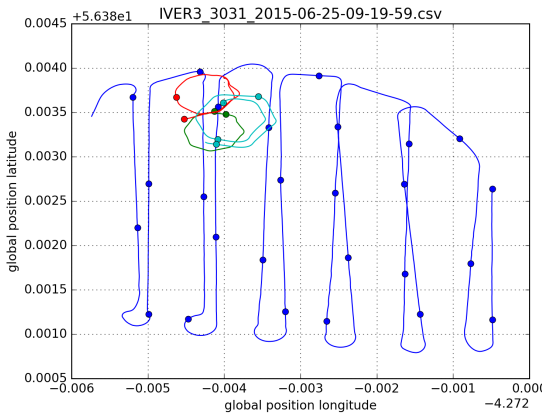
A.1.1 Trajectory Segmentation Examples



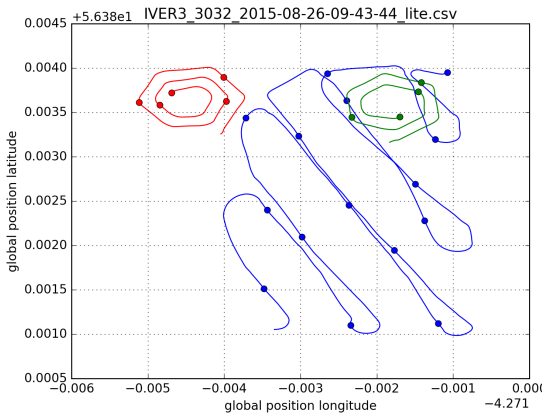
(a) 15t



(b) 15t



(c) 100t



(d) 100t

Figure A.1: Example of two survey missions depicting both types of trajectory patterns: *Lawnmower* and *Spiral*.

A.1.2 Model Candidates Parametrizations

	Input Layer	Batch Size	Optimizer
Search 0	32	16	adam
Search 1	64	64	adam
Search 2	16	16	adam
Search 3	32	64	rmsprop
Search 4	64	16	rmsprop
Search 5	32	64	adam
Search 6	8	32	rmsprop
Search 7	4	64	adam
Search 8	4	32	adam
Search 9	8	64	adam

Table A.1: Table lists output parameters from the random search for the LSTM networks.

	Input Layer	Hidden Layer	Batch Size	Optimizer
Search 0	32	16	adam	
Search 1	64	64	adam	
Search 2	16	16	adam	
Search 3	32	64	rmsprop	
Search 4	64	16	rmsprop	
Search 5	32	64	adam	
Search 6	8	32	rmsprop	
Search 7	4	64	adam	
Search 8	4	32	adam	
Search 9	8	64	adam	

Table A.2: Table lists output parameters from the random search for the LSTM+ networks.

	Kernel	Filters	Input Layer	Batch Size	Optimizer
Search 0	128	128	8	64	rmsprop
Search 1	256	128	64	32	adam
Search 2	32	32	16	16	adam
Search 3	64	256	64	32	adam
Search 4	256	64	64	64	rmsprop
Search 5	64	32	4	16	adam
Search 6	128	128	8	64	adam
Search 7	256	64	32	32	adam
Search 8	256	32	16	16	rmsprop
Search 9	256	32	32	16	rmsprop

Table A.3: Table lists output parameters from the random search for the CNN-LSTM networks.

	Kernel	Filters	Input Layer	Hidden Layer	Batch Size	Optimizer
Search 0	32	64	8	3	32	rmsprop
Search 1	256	256	8	2	32	adam
Search 2	64	64	8	2	32	rmsprop
Search 3	128	32	4	2	64	adam
Search 4	256	256	4	3	64	rmsprop
Search 5	32	64	4	2	16	adam
Search 6	64	256	4	3	32	rmsprop
Search 7	32	32	8	2	64	rmsprop
Search 8	64	32	4	2	64	rmsprop
Search 9	32	128	4	2	64	adam

Table A.4: Table lists output parameters from the random search for the CNN-LSTM+ networks.

	Kernel	Filters	Input Layer	Hidden Layer	Batch Size	Optimizer
Search 0	64	128	32	32	32	rmsprop
Search 1	64	64	4	32	32	rmsprop
Search 2	256	32	4	64	64	rmsprop
Search 3	64	128	64	32	64	rmsprop
Search 4	64	64	32	32	32	adam
Search 5	32	256	64	64	32	adam
Search 6	32	32	4	64	64	adam
Search 7	32	32	16	16	16	rmsprop
Search 8	32	128	4	64	64	adam
Search 9	256	32	4	8	16	adam

Table A.5: Table lists output parameters from the random search for the CNN-LSTM+ networks (ver2).

A.1.3 Network Performance Results

	Search	Accuracy(%)	F1 Score(w)	F1 Score($cl0$)	F1 Score($cl1$)	Total Params
LSTM	0	nweights	92.94	0.9289	0.9593	0.7321
LSTM	1	nweights	92.47	0.9201	0.9573	0.68
LSTM	2	nweights	93.88	0.9393	0.9646	0.7759
LSTM	3	nweights	91.53	0.9204	0.9497	0.7313
LSTM	4	nweights	79.76	0.8258	0.8705	0.5376
LSTM	5	nweights	86.12	0.8013	0.9254	0.0
LSTM	6	nweights	86.82	0.8816	0.9195	0.6364
LSTM	7	nweights	86.59	0.8036	0.9281	0.0
LSTM	8	nweights	86.59	0.8036	0.9281	0.0
LSTM	9	nweights	91.06	0.9141	0.9475	0.6984
LSTM+	0	nweights	85.88	0.827	0.9225	0.2105

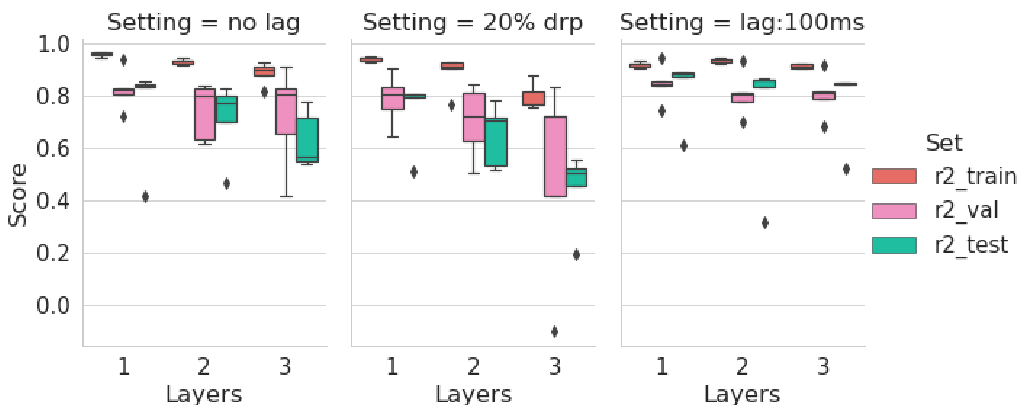
LSTM+	1	nweights	86.59	0.8036	0.9281	0.0
LSTM+	2	nweights	86.59	0.8036	0.9281	0.0
LSTM+	3	nweights	89.41	0.9008	0.9371	0.6667
LSTM+	4	nweights	88.0	0.8385	0.935	0.2154
LSTM+	5	nweights	86.59	0.8036	0.9281	0.0
LSTM+	6	nweights	91.53	0.9186	0.9503	0.7143
LSTM+	7	nweights	86.59	0.8036	0.9281	0.0
LSTM+	8	nweights	86.59	0.8036	0.9281	0.0
LSTM+	9	nweights	91.06	0.9169	0.9466	0.7246
LSTM+-FCN	0	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	1	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	2	nweights	90.59	0.9011	0.9465	0.6078
LSTM+-FCN	3	nweights	90.82	0.9086	0.9469	0.6609
LSTM+-FCN	4	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	5	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	6	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	7	nweights	92.71	0.9256	0.9582	0.7156
LSTM+-FCN	8	nweights	86.59	0.8036	0.9281	0.0
LSTM+-FCN	9	nweights	88.24	0.884	0.9317	0.5763
LSTM-FCN	0	nweights	92.47	0.9272	0.9559	0.7419
LSTM-FCN	1	nweights	90.82	0.9031	0.9479	0.6139
LSTM-FCN	2	nweights	90.82	0.9086	0.9469	0.6609
LSTM-FCN	3	nweights	90.59	0.8923	0.9476	0.5349
LSTM-FCN	4	nweights	92.47	0.9263	0.9562	0.7333
LSTM-FCN	5	nweights	92.47	0.9268	0.956	0.7377
LSTM-FCN	6	nweights	92.24	0.9252	0.9545	0.736
LSTM-FCN	7	nweights	83.76	0.7894	0.9117	0.0
LSTM-FCN	8	nweights	86.35	0.8245	0.9256	0.1714
LSTM-FCN	9	nweights	90.35	0.9101	0.9425	0.7007
CNN-LSTM	0	nweights	89.65	0.8731	0.9432	0.4211
CNN-LSTM	1	nweights	92.0	0.9125	0.955	0.6383
CNN-LSTM	2	nweights	93.65	0.9392	0.9627	0.7874
CNN-LSTM	3	nweights	94.12	0.9405	0.9662	0.7748
CNN-LSTM	4	nweights	90.82	0.898	0.9486	0.5714
CNN-LSTM	5	nweights	91.29	0.9043	0.9511	0.6022
CNN-LSTM	6	nweights	91.53	0.9171	0.9507	0.7
CNN-LSTM	7	nweights	91.29	0.9126	0.9498	0.6726
CNN-LSTM	8	nweights	93.88	0.9401	0.9644	0.7833

CNN-LSTM	9	nweights	91.53	0.9212	0.9494	0.7391
CNN-LSTM+	0	nweights	88.24	0.8926	0.929	0.6575
CNN-LSTM+	1	nweights	92.0	0.9194	0.9539	0.6964
CNN-LSTM+	2	nweights	91.76	0.916	0.9528	0.6789
CNN-LSTM+	3	nweights	91.53	0.92	0.9499	0.7273
CNN-LSTM+	4	nweights	89.18	0.8693	0.9404	0.4103
CNN-LSTM+	5	nweights	92.24	0.9215	0.9553	0.7027
CNN-LSTM+	6	nweights	92.24	0.9164	0.9562	0.6598
CNN-LSTM+	7	nweights	91.76	0.9139	0.9531	0.6602
CNN-LSTM+	8	nweights	89.18	0.8918	0.9375	0.5965
CNN-LSTM+	9	nweights	89.65	0.8784	0.9427	0.4634
CNN-LSTM+2	0	nweights	92.71	0.9256	0.9582	0.7156
CNN-LSTM+2	1	nweights	92.47	0.9216	0.9571	0.6923
CNN-LSTM+2	2	nweights	88.0	0.8385	0.935	0.2154
CNN-LSTM+2	3	nweights	87.53	0.8353	0.9323	0.209
CNN-LSTM+2	4	nweights	88.24	0.8487	0.9359	0.2857
CNN-LSTM+2	5	nweights	89.41	0.8997	0.9374	0.6565
CNN-LSTM+2	6	nweights	90.59	0.9101	0.9446	0.6875
CNN-LSTM+2	7	nweights	94.59	0.9453	0.9689	0.7928
CNN-LSTM+2	8	nweights	91.06	0.9077	0.9489	0.6415
CNN-LSTM+2	9	nweights	92.0	0.9231	0.953	0.7302

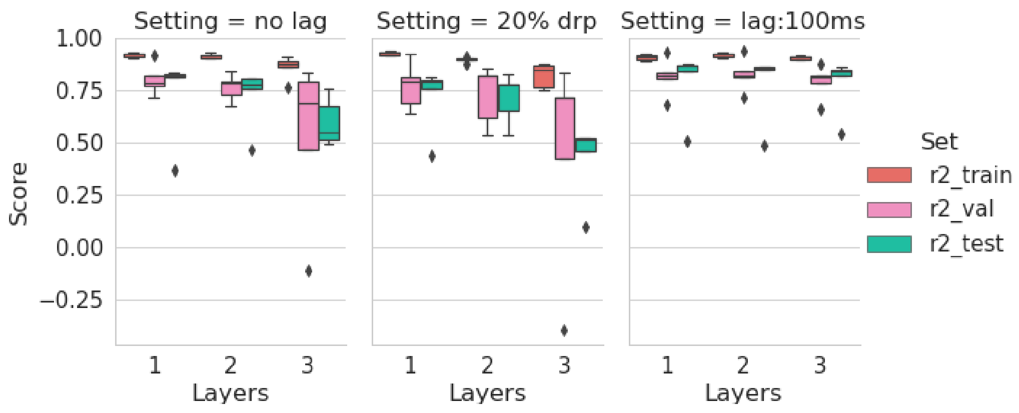
Table A.6: Non class-weighted results summary.

A.2 Chapter 6

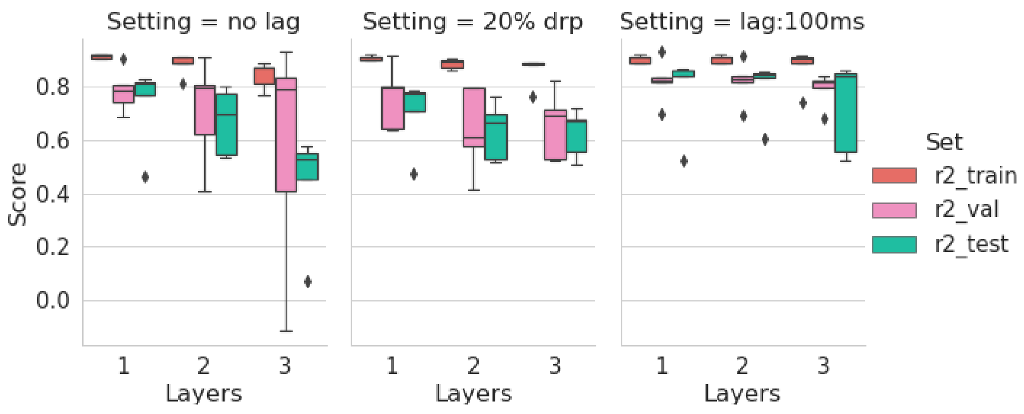
A.2.1 Performance by CV Sets



(a) IO Sequence Length = 25ts/250ms



(b) IO Sequence Length = 50ts/500ms



(c) IO Sequence Length = 100ts/1s

Figure A.2: Categorical Boxplots of the best HBBO tuned networks. Each individual plot shows the R^2 score across the Cross-validation sets for each Encoder-Decoder depth for a specific non-HBBO setting.

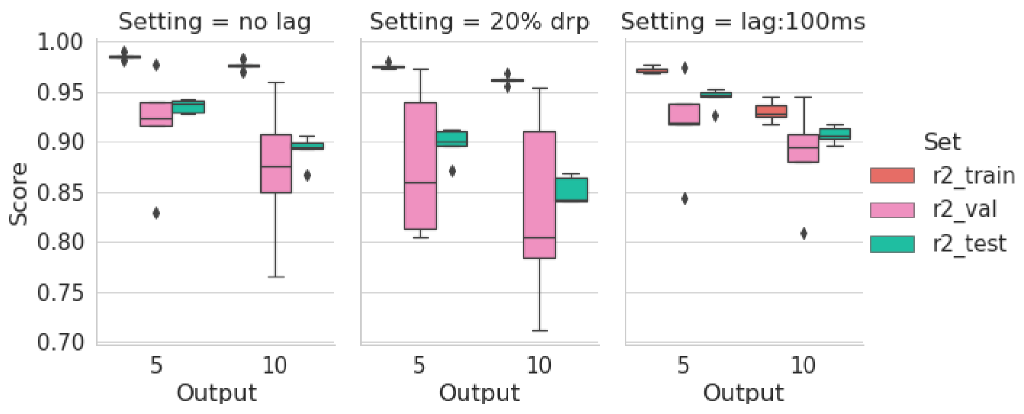


Figure A.3: Categorical boxplots for variable output experiments showing scores in the CV process