Using Machine Learning Methods to Aid Scientists in Laboratory Environments

by

Rory Coles

A thesis submitted to the
School of Graduate and Postdoctoral Studies
in partial fulfilment of the requirements for the degree of

Masters of Science

in

Modelling and Computational Science

Faculty of Science

Ontario Tech University

Oshawa, Ontario, Canada

December 2019

Thesis Examination Information

Submitted by: **Rory Coles**

Masters of Science in **Modelling and Computational Science**

Thesis Title: Using Machine Learning Methods to Aid Scientists in Laboratory Environments

An oral defense of this thesis took place on Dec 5, 2019 in front of the following examining committee:

Examining Committee:

Chair of Exammining Commitee	Franco Gaspari
Research Supervisor	Lennaert van Veen
Research Co-Supervisor	Isaac Tamblyn
Examining Committee Member	Faisal Qureshi
Thesis Examiner	Yuri Grinberg, NRC

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

As machine learning gains popularity as a scientific instrument, we look to create methods to implement it as a laboratory tool for researchers. In the first of two projects, we discuss creating a real-time interference monitor for use at a radio observatory. We show how deep neural networks can be used to assist with the detection of radio-frequency interference around the site, and consider methods of unsupervised learning to identify patterns in the detections. In the second project, we show how a reinforcement learning agent can build an internal hypothesis of its environment, using experience from past measurements, that it can then act on. We demonstrate how our newly developed method can be used to learn the dynamics of physics-based models and exploit the knowledge gained to achieve a given objective with measurable confidence. We also demonstrate how the agent's behaviour changes when the frequency of certain measurements is limited.

Keywords: Machine Learning; Deep Learning; Neural Networks; Reinforcement Learning; Bayesian Modelling;

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorise the University of Ontario Institute of Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorise University of Ontario Institute of Technology to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Statement of Contributions

All text and figures are produced by Rory Coles, except where otherwise stated. The work described in Chapters 2 and was performed with the Herzberg Astronomy and Astrophysics Research Centre under the primary supervision of Stephen Harrison. The algorithms shown in Chapters 2 and was developed by Rory Coles using the TensorFlow and scikit-learn packages respectively. The survey and cumulant data used in this work was provided by Stephen Harrison. Figures 2.1 and 2.10 were produced by Stephen Harrison. In Section 2.5.1 the work discussed under the subheading "Adding Expert Knowledge" is being conduction by Stephen Harrison, David Del Rizzo, and Timothy Robishaw. The work discussed under the subheading "Autoencoders" is being conducted by Nicholas Bruce. Rory Coles is conducting the work discussed under the subheading "Subspace Clustering". The work described in Chapters 3 was performed at the Security and Disruptive Technologies Research Centre under the primary supervision of Isaac Tamblyn. The original CartPole, MountainCar, and Acrobot environments were used from the OpenAI gym package. The software for all the reinforcement learning optimisation algorithms presented in Chapter 3 were written by Rory Coles. The software for the neural network and gaussian process models used in Chapter 3.3 were written by Rory Coles using the open source packages TensorFlow and GPy respectively. All of the environments presented in Chapter 3.3 were created by Rory Coles.

Contents

List of Figures

Abbreviations

A3C Asynchronous Advantage Actor-Critic.

AI Artificial Intelligence.

ANN Artificial Neural Network.

CHIME The Canadian Hydrogen Intensity Mapping Experiment.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DNN Deep Neural Network.

DRAO Dominion Radio Astrophysical Observatory.

EA Evolutionary Algorithm.

EDNN Extensive Deep Neural Network.

ES Evolutionary Strategies.

GA Genetic Algorithm.

GP Gaussian Process.

IOU Intersection Over Union.

MC Dropout Monte-Carlo Dropout.

MDP Markov Decision Process.

ML Machine Learning.

MNIST Modified National Institute of Standards and Technology.

NEAT NeuroEvolution of Augmenting Topologies.

OPTICS Ordering Points To Identify the Clustering Structure.

POMDP Partially Observable Markov Decision Process.

PreDeCon subspace PREference weighted DEnsity CONnected clustering.

RBF Radial Basis Function.

ReLU Rectified Linear Units.

RFI Radio Frequency Interference.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SK Spectral Kurtosis.

SL Supervised Learning.

SVM Support Vector Machine.

TWEANN Topology and Weight Evolving Artificial Neural Network algorithms.

UL Unsupervised Learning.

YOLO You Only Look Once.

1 Introduction

In recent years, one of the most exciting tools in data science and Artificial Intelligence (AI) that is being developed by researchers is Machine Learning (ML). This collection of algorithms have been shown to surpass human abilities in a variety of tasks such as image classification [1], language processing [2], and video games [3]. As a result, ML is becoming commonplace in everyday life, changing the way we interact with many devices and services. For example, numerous companies have adopted some form of ML for tracking consumer preferences [4], many websites detect anomalous behaviour to decrease security risks [5], and AI-enabled predictive text can be used in many online conversations and search engines [6].

Data science has always been popularly implemented in the scientific community, and ML is no different. Researchers who specialise in ML explore the development of algorithms and their applications and the applications of ML can be seen in a broad range of fields of science. For example, ML methods have been used for molecule design [7, 8], analysing medical X-Rays [9], and tracking wildlife behaviours in the Serengeti [10]. In physics, ML has been used to simulate structure formation of the universe in a fraction of the time of an N-body simulation [11], predict phase transitions [12], and to optimise the efficiency of heat engines [13].

The recent increase in the popularity of ML is, in part, because of new algorithms, increased computational power, and large datasets all becoming available.

These factors allowed for deep learning to become feasible. Deep learning makes use of Artificial Neural Networks (ANNs) that can build up a hierarchical understanding of the world [14]. Complex concepts are understood by building upon layers of simple concepts. Several layers of concept representations often need to be learnt, and therefore this is referred to as deep learning. ANNs are a recurring tool throughout this work, and so we will discuss them in Section 1.1.

Most ML algorithms and problems can be broadly separated into three main fields: Supervised, Unsupervised, and Reinforcement Learning. Supervised Learning (SL) algorithms aim to learn a relationship between a dataset of measurements and target variables corresponding to them [14]. If a representative relationship can be successfully found, then the algorithm can be used to predict the target variable for previously-unseen data. In the past, traditional model-fitting techniques have been struggled to map diverse complex and non-linear relations. However, modern techniques are designed for these kinds of relations and have excelled at mapping them.

The second category of algorithms is Unsupervised Learning (UL), a collection of tools that are primarily used for creating representations of datasets [15]. These algorithms take only the unlabelled data as inputs and seek to learn structures of the data that may not be obvious to the user. As such, there is no absolute truth that is associated with the learning process. Almost all UL algorithms are learning a probabilistic model of the dataset [15]. UL algorithms are commonly used for dimensionality reduction or grouping data points together with similar points in the dataset.

Lastly, we have Reinforcement Learning (RL), the study of behavioural rewards while acting in a set environment [16]. RL algorithms produce a sequence of actions that affect the state of the environment. These actions induce a scalar value which acts

as a reward or punishment, depending on if the action was beneficial or detrimental. The goal for the algorithm is to maximise the reward it receives from the environment over some period of time, and ultimately it should be able to respond to situations it hasn't seen before. RL doesn't require a dataset of the same form that is seen with the other two categories. Instead, the training examples are generated through experience from acting in the environment.

1.1 Artificial Neural Networks

In all three of the described subfields of ML, a large number of successes have come with the use of ANNs. These artificial networks are computational systems inspired by biological neural networks in the brain that act as a framework for ML algorithms. An ANN is comprised of a collection of nodes, known as neurons, joined together by connections, known as weights [14]. These weights can be adjusted to alter the signal strength that passes through the respective connection. ANNs are typically organised into layers which can each perform different operations on their inputs. In a feed-forward network, the signal passes from the input layer through a hidden layer to an output layer. The input layer is of the same dimension as the input data, likewise for the output layer and the expected output size. The hidden layer can be altered to have as many nodes as is necessary to approximate the required function. The input to any neuron beyond the input layer is given by:

$$I_j = \sum_{i=1}^{n} S_i W_{ij} \tag{1.1}$$

where n is the number of input nodes, S_i is the output signal from the connected nodes in the previous layer, and W_{ij} is the corresponding weight. The output signal

is then given by:

$$S_j = f(I_j + b_j) \tag{1.2}$$

where b_j is a bias associated with neuron j, and f is the activation function. The bias is an additional parameter in the ANN that is equivalent to the intercept that is added to a linear equation. It is a constant that adjusts that the weighted sum of the neuron's inputs, which helps the network create an accurate model. The activation function is a non-linear function that alters how the output of the node is perceived. The function decides if that particular neuron activates or not and by adding non-linearities the ANN can map complex functions with a relatively small number of neurons. We will discuss activation functions later in this chapter.

Deep Neural Networks

ANNs are not a new concept; they've been around for decades [17] but have only become viable with recent advancements in the field, as these allowed for Deep Neural Networks (DNNs) to be developed. What distinguishes a DNN is that there are multiple hidden layers which allow for these systems to learn hierarchical features and build upon the previous layer [14]. The depth of a DNN is not strictly limited; there have been examples where over 100 layers have been used [18]. Adding additional depth to the network allows for more complex non-linear relationships to be modelled, however, does come with additional challenges. The number of layers is often limited by computational time. Increasing depth increases the number of calculations and the number of parameters to optimise.

Once the network architecture, i.e the number of layers, the width of each layer, and the activation functions, has been determined, the ANN can be represented as a black box system with two methods. The first method makes a prediction using

the current internal parameters of the network. An input is passed into the ANN, and the prediction is the most likely result for that input based on the past training experience. This leads on to the second method, training the ANN. The training process uses both the predicted and expected outputs to tune the weights such that the two output values are as similar as possible. The primary technique for tuning the weights to train an ANN is gradient descent.

Gradient descent makes use of the instantaneous rate of change in the relationship between internal parameters of the ANN and a pre-defined loss function. The loss function calculates the error, E, between the predicted output and the desired output. When optimising an ANN, we are altering the parameters of the ANN to minimise the error. Knowing how changing a particular parameter will change the loss function, the weights can each be tuned to minimise the error. As the ANN has many parameters, we need to measure the partial derivative of each parameter's contribution to the total change in error. At this point with a single layer ANN, we would update the weights until the loss has reached a minimum, where the derivative is 0. An individual weight update takes the form of Equation 1.3. α is a scaling factor between 0 and 1 that alters the step size of the weight update, known as the learning rate.

$$w_{new} = w - \alpha \frac{\partial E}{\partial w} \tag{1.3}$$

With a DNN, we have multiple layers of weights that process the input signal sequentially. To update the weights in DNNs, we need to backward propagate information about the error through the entire network so it can alter all the weights. This algorithm is known as backpropagation [19]. As with other gradient descent methods, backpropagation makes use of the differentiation of the loss function to guide tuning the weights. The goal is simple: to adjust each weight proportionally to how much

it contributes to the overall error. To do so, during the forward pass of the network, as the signal passes through each layer, we create a stack of the function calls and their parameters. During the backward pass, as each function call is de-stacked, back-propagation establishes the relationship between the error and the parameters in that layer through the chain rule of calculus. This allows the DNN to obtain the derivative rate for each weight which, when multiplied by the learning rate, gives the weight update value. This process is done iteratively until the loss function converges.

Activation Functions

Consider the output of a single neuron, Equation 1.2. If we were to set $f = c \cdot x$, where c is a constant with a magnitude greater than 0, the output signal of the neural network would be a linear function. If an ANN only has linear functions or no activation function at all, then any amount of layers can be replaced by a single linear function. Any combination of linear functions in a layered manner is still a linear function. Thus only having linear functions in the ANN does not provide it with the ability to approximate non-linear relationships. To prevent this problem, we need to add non-linearities into the ANN through the activation functions. There are additional problems with using only linear functions that we will discuss before continuing. The derivative of a linear function $f = c \cdot x$ with respect to x is a constant c. If the training method for an ANN uses a gradient descent based method, then the weight updates will be constant and not depend on any change in the input.

One of the most popular and simple activation functions is uses Rectified Linear Units (ReLU) (Eq.1.4) [20], shown in Figure 1.1a. The ReLU function gained popularity due to how simple and efficient the calculation is, which keeps training and inference times as short as possible, while still allowing the ANN to approximate

complex functions.

$$f(x) = max(0, x) \tag{1.4}$$

The downside of using the ReLU function is that it is possible for a neuron in the network to get stuck, known as the dying ReLU problem. This occurs if a large weight update causes the neuron to output a negative value. At this point, the output and the gradient would both be 0, and so the neuron will not change during gradient descent learning. As a result, the neuron may never activate across the entire training dataset and thus be useless. This can often be mitigated by limiting the value of the learning rate, preventing the large weight updates that may cause this problem. Another solution is to have a small positive gradient on the negative inputs to provide those neurons with a chance to recover. This is known as using Leaky ReLUs [21], shown in Figure 1.1b.

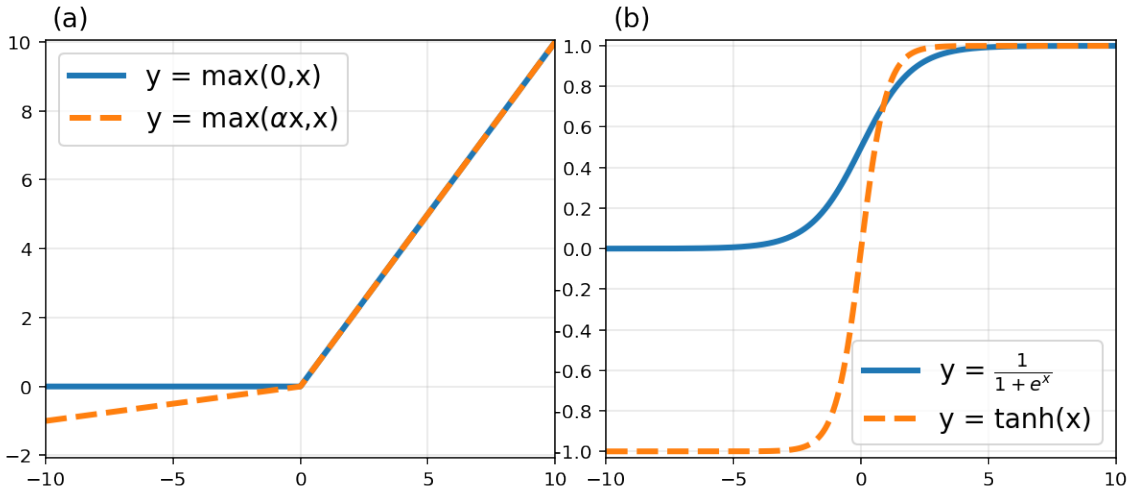

Figure 1.1: Activation functions in the range [-10,10]. (a) Shows the ReLU and Leaky ReLu functions with $\alpha = 0.1$. (b) Shows the Sigmoid and tanh functions

There are also activation functions that bound the output of the neurons. For example, the logistic function is another commonly used activation function that bounds the output of a neuron in the range [0,1] [22]. This range is especially useful

if the model is predicting a probability value. Also known as the sigmoid function, the form is as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

By its nature, the sigmoid function has a high gradient in the range [-2,2]. Weight updates using these gradients will, therefore, cause distinct changes in the weight values. On the other hand, this function also experiences the vanishing gradients problem. When the input to the activation function is either very high or very low, the gradient is extremely small. As such, the weight update will not make significant changes, and the network will train very slowly or stagnate completely.

An alternative to the sigmoid activation function is the tanh function, which is a scaled sigmoid function. This function is bounded between [-1,1] and is very similar to the sigmoid function in behaviour, including the vanishing gradient problem. The main benefit of using the tanh function is for the same reason that the inputs should be normalised; because this function is more likely to produce outputs that are close to zero on average.

1.2 Outline of Thesis

The work done in thesis focuses on utilising ML techniques to supplement scientific researchers in laboratory environments. This work is split into two major projects. The first project consists of the work done in Chapters 2 and 2.3.4 with the goal of detecting and categorising interference signals. The second project consists of the work Chapters 3 and 3.3. This work focuses on studying RL methods and applying them to various problems of interest.

In Chapter 2, we will introduce the concept of SL and how ANNs can be used to locate objects in images. We will showcase how SL algorithms were used to make a

signal interference monitor that is now in use at a radio observatory. While many steps had been taken around the observatory to reduce interference, the staff wanted a real-time monitoring system to assist workers on-site in mitigating the interference. We will describe the process of making the tool that detects interference in real-time and testing it on incoming data.

In Chapter 2.3.4, we will discuss how UL can be used to identify patterns in the interference signals detected around the observatory site. A secondary goal of the observatory project was to identify previously unknown patterns in the detections so that groups of detected signals can be studied together. To do so, we investigated clustering methods that can be used to identify similar signals, but more importantly signals which do not belong to a group. We will also show how we can visually explore data in high-dimensional spaces.

Then in Chapter 3, we will begin the second project. We will discuss RL algorithms, particularly evolutionary methods, and what can be done to improve training time for RL optimisation. We will compare selected algorithms and show that evolutionary methods are a viable alternative to traditional RL optimisation methods. We will also present some further techniques for improving the training time of RL algorithms.

Finally, the Chapter 3.3, we will present a new paradigm for RL agents that can build an internal hypothesis of their environments. The agent must use experience from past measurements to create their hypothesis which will contain some uncertainty measure. Therefore, the agent must learn to act in an environment when there is uncertainty in the state and when to take detailed observations to update the model appropriately. We will show how this newly developed method can be used to learn the dynamics of physics-based models and exploit the knowledge gained to achieve a given objective with a measurable confidence.

2 Real-Time Signal Detection and Classification

In the past, Radio Frequency Interference (RFI) monitoring at Dominion Radio Astrophysical Observatory (DRAO) has been focused on protected wavebands, but now with the observing bandwidth increasing, the site needs to limit sources of RFI. Previous attempts at monitoring RFI have faltered due to the volume of data produced and limited labour-hours available to study it. To expand on operating capabilities, the staff at DRAO want a sophisticated interference monitoring system to be established. The goal for this system is to be able to detect and categorise terrestrial signals around the site with little to no human interaction. By doing so, we can characterise the RFI scene around the site, study populations of events together, and identify if a signal is novel compared to standard events.

The following sections will first provide the necessary background for the problem and the proposed tools to solve it. We will then discuss how the first prototype for this system was developed, specifically the detection of RFI signals in real time. Finally, the current limitations will be discussed, as well as the proposed next steps to be taken.

2.1 Radio Frequency Interference

Since the first detection of radio waves originating from outside of Earth, the study of extraterrestrial radio frequencies has been essential to our understanding of the universe. Observations of astronomical objects such as stars and galaxies identified these objects as sources of radio emission [23–25]. Objects that were unclassified at the time, such as radio galaxies and pulsars, were also found to be sources [26–29], and cosmological studies allowed scientists to discover the cosmic background radiation [30–32]. In recent years, the radio band has become an increasingly promising waveband for cosmology observations and the portion of the spectrum that is being monitored is increasing [33–39].

While radio astronomy is a growing field of science, it is competing for operating space with modern life. Cell-phones, GPS-transmitters, radio and television broadcasting etc. all operate in the radio spectrum, as assigned by the Canadian Table of Frequency Allocations [40]. Radio astronomy has some protected frequencies to operate in, such as between 73-74.6 MHZ and 406.1 - 410 MHz for solar wind and pulsar observations respectively [41]. However, for some projects this is not enough and scientists must contend with terrestrial signals, such as those listed above, interfering with those that are extra-terrestrial. Signals such as these are known as RFI. To deal with these signals, observatories are often located in remote locations and electrical equipment that could cause interference is either prohibited or must be adequately shielded.

This is not a new problem; many methods of post-processing the data have been developed to detect and mitigate RFI, with the goal of recovering valuable information from contamination. If a specific type of signal is being searched for, a matched filter will often be used [42–44]. This archetype of detection algorithms will often have to

balance the probability of detecting specific RFI signals and the probability of giving a false positive, in other words, classifying RFI-free data that isn't contaminated. However, having a specific filter for each type of source is impractical for RFI searching as there is a wide variety of sources that could be detected, many of which may not be known beforehand. Therefore, an RFI detection algorithm needs to be robust in detecting a variety of signals. One such test that has shown to be effective is known as Spectral Kurtosis (SK). An SK test determines whether a section of the observed signals differ from the thermal noise [45–50]. Thermal noise is a major source of background noise that appears in electronic circuits such as receiver input circuits that takes a Gaussian form. The statistical properties of this distribution will differ when a non-Gaussian signal is observed, indicating RFI is present.

RFI signals tend to have a received power many orders of magnitude higher than astronomical observations. This means if an RFI signal overlaps with important data, very little can be done to recover the original data. As a result, contaminated data is often just removed in a process known as Excision. This can be done either through deleting the contaminated data or replacing it with Gaussian noise similar to the thermal noise [50–52].

Ultimately, a telescope is the best RFI detector. However, information from telescopes can take a while to be processed; sometimes the data is not available until the entire observation is complete. To reduce the amount of data lost from RFI, there is a need for proactive monitoring to limit the amount of RFI that would be detected by telescopes. If we know that there is a device nearby that is emitting RFI, we can ensure that it is turned off as soon as possible. Doing so would shorten the overall transmission time that the device is emitting RFI for compared to if it went undetected.

2.1.1 Dominion Radio Astrophysical Observatory

At the DRAO, new telescopes such as The Canadian Hydrogen Intensity Mapping Experiment (CHIME) [53] require the observatory to maintain a clean spectrum to the extent possible from 400 MHz to 2 GHz; however, most of this band is not protected for radio astronomy. While DRAO is located in an area of low population density and surrounded by hills, the site is still located within 2km of a golf course, 5km of cellular towers, and 20km from the city of Penticton that hosts a regional airport; all of which contribute to the radio frequency environment at the observatory. In spite of this, one of the larger contributions of RFI comes from on-site; either electrical equipment required for observatory operations or unintentionally from staff, such as leaving their mobile phones on while at work. From this, we can see two broad categories that the RFI signals will belong to; those which are from sources that the observatory can't reasonably prevent, such as the cellular towers and planes overhead, and those which can be prevented, like cell phones.

Previous attempts to monitor interference at DRAO have either been limited in scope or produced too much data that can reasonably be examined at a site with limited labour-hours to commit. The latter results in large amounts of data being stored but not investigated. As a result, RFI is often first detected when it shows up in the data received from the telescopes, which is far too late. At this point, the researcher could perhaps go back and look through the data from the site monitor to identify the RFI, but the data is already contaminated and so is often just mitigated in the results.

Instead of dealing with the RFI signals in the data, we look to remove it at the source. We want a system that can monitor the radio spectrum, handle the large volume of data produced by a receiver, and assist observatory staff in both

identifying and finding the sources of the RFI signals. To turn this system into a more sophisticated monitor, it needs to do several tasks. The tool should provide a continuous characterisation of the radio frequency signals that are around the site. Having access to information that characterises the RFI around the site, such as the centre frequency, bandwidth, time of day, etc., is extremely valuable to researchers when planning observations. Additional information such as received power, channel parameters, and modulation parameters are also helpful when identifying the device causing the RFI. All of the information gathered from the monitor should be stored in a database for future analysis and exploration. To further this, we would like this system to recognise behaviours in the data that may not have been previously identified. Doing so will aid in characterising what is normal around the site and help plan observations. Finally, all the above needs to be able to run as autonomously as possible while providing near real-time alerts of new behaviours observed at the site: changes in patterns, new signals, etc. A system such as this will allow the staff at DRAO to expand the monitoring program with little change to current personnel resources.

An omnidirectional receiver was installed above the main office building at DRAO to receive terrestrial radio signals from around the site . This receiver is much less sensitive than the telescopes on site, therefore any signal that can be detected by this receiver would be many orders of magnitudes more powerful than astronomical signals and thus we can say it is interference. Figure 2.1 shows a snapshot of the signals received from the receiver. This figure is known as a waterfall plot and it presents frequency and time data. The intensity at one frequency-time point indicates the received power on that frequency channel at that time. We can see high intensity areas at two separate frequencies. Both signals are narrowband which means they are covering only a small frequency range. The signal closer to 458.5MhZ is continuously

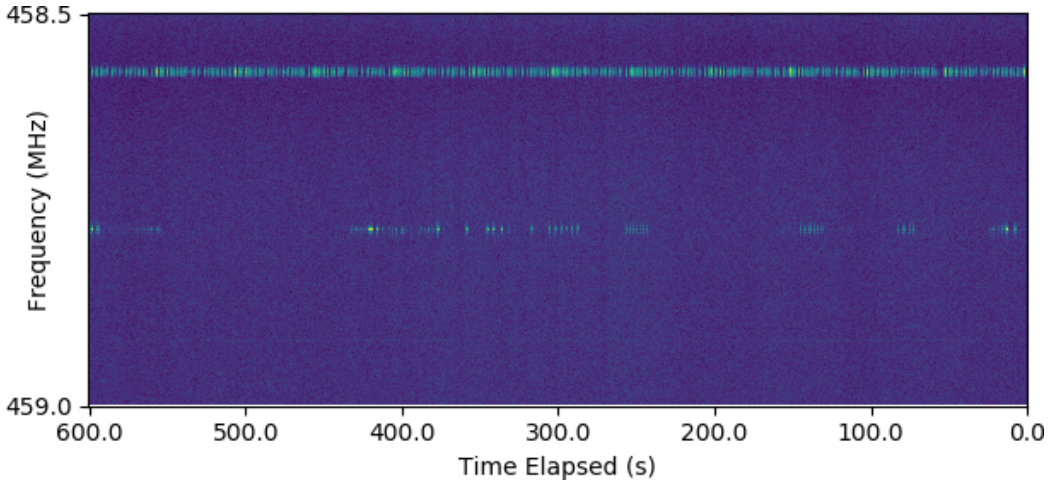

Figure 2.1: A waterfall plot of an example RFI signal over a 10 minute time period.

on, even beyond this time frame, therefore is known as a static signal. On the other hand, a signal that is only detected for a short period is known as a transient signal. The static signals, such as from cellular towers, are always active and so they are easier to detect and mitigate compared to transient sources. A waterfall plot presents the data in a two-dimensional format, similar to an image. A signal seen in the waterfall plot is a continuous area of high pixel intensity, similar to an object in an image. However, given that one of the dimensions is time, a waterfall plot is also similar to a video showing the recent history of detections.

The receiver directly provides information about the power, time, and frequency of any signals it receives, therefore, we would like our system to be able to detect interference using just this information. Doing so keeps the number of operations performed on the raw data low, therefore allowing for quick detection of the signals. Once detected, the signals can be extracted and then further analysed. In addition, the time periods and frequency channels where RFI is most often detected is some of the most valuable information we could provide observatory staff. As we saw in Figure 2.1, we can identify RFI as areas of high pixel intensity. To detect these areas in the image, we look to an area of ML known as supervised learning.

2.2 Supervised Learning

To automatically detect RFI events, such as those seen in Figure 2.1, we will being using methods from SL; the branch of ML that focuses on predicting an output from input using knowledge from previous examples [14]. Each of these examples is a pair that includes an input, x_i, and a corresponding true output, y_i, known as a label, such that the training set takes the form $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ assumed to be obtained from some fixed distribution. An SL algorithm would be trained on many of these examples to produce a function, f, that can be used to map an input to output, such that $y_i = f(x_i) + \epsilon$, where ϵ is some independent error term. An optimally trained model will be able to generalise from the training set and make reasonable predictions on unseen data.

For example, let us consider the case of categorising handwritten digits. This is a popular and well-explored example in the field of ML due to the Modified National Institute of Standards and Technology (MNIST) dataset [54,55]. The MNIST dataset consists of handwritten digits that has been made available for anyone wishing to test learning algorithms on real-world data. In this case, the training set would be thousands of pictures of these digits with the label representing the correct number shown by the image. An algorithm would have to learn a maps from an image in the dataset to a number, and also apply that model to new images that do not have labels. This example is a common introduction to SL, yet has been explored to great length in the literature [56–59] and similar datasets have been released in the same style [60–62].

SL problems can be split into two tasks: regression and classification. The example above regarding assigning hand-written digits a number from a finite selection is classification. The goal of a classification model is to assign each input a label from

a list of discrete possibilities. An unseen input will be assigned to the class it most likely belongs in based on what the model has seen during training. Regression, on the other hand, is the task of predicting a single continuous value based on the inputs. A typical example often demonstrated is predicting house prices. There are datasets available, for example, the Boston Housing Dataset [63,64], that have information on various characteristics of houses and the surrounding area such as crime rate, access to highways, pupil-teacher ratio, and the value of the house. An ANN can be trained on all the features, with the true label being the price, and obtain a model to predict the price given the same set of features.

There is a range of SL models that have been devised, and each one has scenarios where it excels and scenarios that it does not. Support Vector Machines (SVMs) [65], for example, are SL models that are most closely associated with binary classification of data. This model utilises a linear hyperplane to split data between two categories. The hyperplane is a subspace that has one less dimension than the feature space. An optimally trained SVM constructs a hyperplane that represents that largest separation between the two classes. For nonlinear classification, the original feature space can be mapped to higher dimensions using a kernel function. This is known as the kernel trick [14] and it enables the model to learn nonlinear functions. SVMs have been shown to classify images and hand-written characters [66–68], and are further being used in medical fields [69, 70]. The SVM method has been extended into further applications, including multi-class classification [71], regression [72], and anomaly detection [73]. SVMs have some downsides though, one of the biggest being the kernel selection is not trivial [74]. The choice of the kernel can be very problem-specific, and often kernels do not generalise well. This is one of the motivations towards using ANNs to predict the mapping function for SL problems. Indeed, Goodfellow states that "the deep learning renaissance began when Hinton et al. (2006) demonstrated

that a neural network could outperform the Radial Basis Function (RBF) kernel SVM on the MNIST benchmark" [14].

The input nodes to the neural network would be all the individual features available to the model, and the output would be the prediction. In the case of classification, the network may have as many outputs nodes as there are categories and the node with the highest value represents the predicted label for that input. For a regression problem, the prediction will be continuous values for the dependent variable in the system. To train the neural network, there needs to be a measure of how the network is performing. In other words, we need a loss function as discussed in Section 1.1. For SL, this loss is obtained by comparing the predicted output from the network, $Y_{predicted}$, to the true label, Y_{true}. As this label is provided in the dataset, the network has a way to measure exactly how good the prediction is. The specific form of loss function will depend on the problem. The goal of this function is to provide a metric of the precision lost if the true output is replaced by the predicted output. A common loss function for regression takes the form of the Mean Squared Error summed over all the examples in the dataset:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(Y_{true} - Y_{predicted})^2 \tag{2.1}$$

This function allows for larger errors to dominate over smaller errors and gives an absolute value so the error is the same regardless of overshooting or undershooting. The goal of the training process is to minimise this loss function by tuning the neural network weights as described in Section 1.1.

Deep supervised learning has helped to revolutionise many fields. Neural networks have been used successfully in natural language processing, Google Translate being an iconic example [75, 76]. Word embedding, such as word2vec [77], is used to transform

words into a unique scalar vector. When used along with Recurrent Neural Networks (RNNs), neural networks which make use of not only the current input but those seen previously, deep learning algorithms can translate entire sentences, including grammar, instead of single words [78–80]. Similarly, speech recognition has benefited from deep learning considerably. Many homes now have smart speakers, such as those from Google or Amazon, that allow consumers to interact with them purely by speaking, and thanks to deep learning [81, 82] these are now accurate enough to be useful in daily life.

These are just some of the successes that deep supervised learning has assisted with. However, it can be a useful tool for any application where the model has access to training data and the corresponding ground truths to be trained against. As a result, there are many subfields of SL; one such that we make use of in particular is computer vision.

2.2.1 Computer Vision

Computer vision is a field of study within AI and SL that focuses on how computers can be used to replicate the tasks that human vision does. Simply put, high-dimensional data comes in as input and the computer needs to be able to extract high-level understanding. As with the previously mentioned applications, deep-learning has transformed computer vision such that it surpassed human ability within a few years of its inception.

One of the biggest applications in popular culture is the use of computer vision in cameras and smartphones [83, 84], particularly with face detection and tracking. Camera filters on social media applications now rely on computer vision to track facial features and alter the image. These alterations could be to change the facial expression, the perceived age of the person in the image, or change the style of the

person. Other applications include AI driver assists [85, 86], sports analysis [87], and geolocalisation on images [88]. This field has itself further branched into many sub-domains such as image restoration, motion estimation, and most relevant to this work, object and event detection.

Objects in a single frame can be bounded by a rectangular box that encompasses the object, known as a bounding box. This box is a set of coordinates that uniquely define the space where the object is located in the image. When a deep learning model is tasked with finding an object in an image, it is learning to predict the bounding box of the object. To compare the prediction against the true boundary boxes, the Intersection Over Union (IOU) metric is commonly used [89]. The IOU is a gauge for how similar two objects are. It is defined as the size of the intersection between the two objects divided by the size of the union. The size of the intersection of two boxes is the area that falls into both the first box as well as the second (Fig. 2.2a), whereas the size of the union is the area covered by both boxes (Fig. 2.2b).

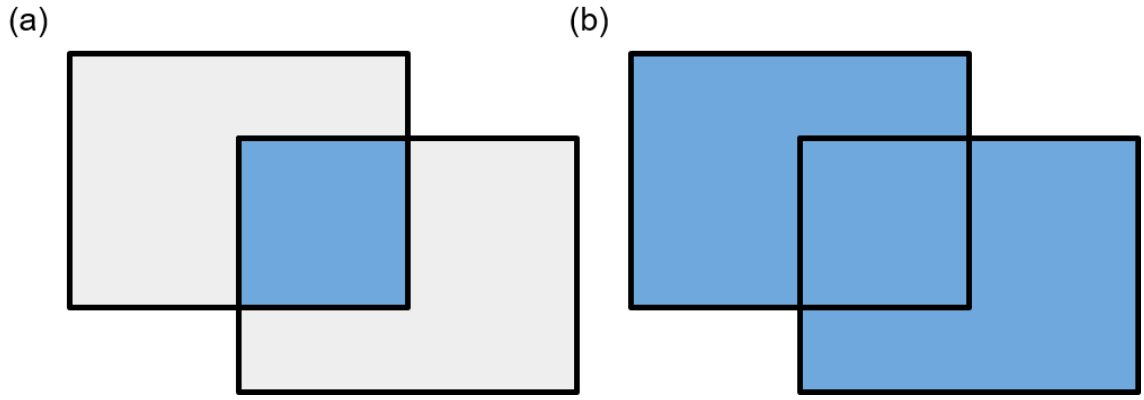

Figure 2.2: (a) The Intersection of Two Rectangles. (b) The Union of Two Rectangles

To search for a single object in an image, an ANN can be trained to predict the coordinates of the bounding box for the image, using the IOU as a measure of performance. However, this methodology will only work for a limited number of cases. If the image is known to always have n sources, then we can train the ANN to learn

n sets of bounding boxes as we have done with $n = 1$. It becomes more difficult if the number of sources is unknown. For example, if the network is trained to obtain five boxes but there is only one true source, we may end up with the remaining boxes giving false positives. On the other side of this, if the network is trained to obtain two boxes, but there are three in the image, then the result will produce a false negative.

One of the more successful object detection algorithms that can detect multiple objects in a frame is known as You Only Look Once (YOLO) [90]. The premise of this algorithm is that the image is split up into an $S \times S$ grid of evenly sized tiles and every tile is passed individually into the same DNN. For each tile, the net can predict a set number, n, of bounding boxes whose centres fall within that grid cell. For each bounding box, the ANN also predicts a confidence score between 0 and 1 which indicates how likely it is that the box contains an object. Therefore given a single image, the algorithm predicts nS^2 bounding boxes along with their confidence scores. A more detailed description and implementation is shown in Section 2.3.

2.3 Real-Time Signal Detection

2.3.1 Single Source Detection

To begin with, a DNN was trained to recognise a single source of interference in time-frequency space based on power levels or pixel intensity; in more general terms, locate a single object in a two-dimensional image. The DNN accepts a two-dimensional input and predicts the coordinates for the signal in the image. These coordinates form a bounding box that encompasses an area of high pixel intensity. This can be formulated as a SL problem by using a training set of these images with their true labels being the coordinates required to define a bounding box.

As aforementioned, a lot of data surrounding RFI has been collected in the past at DRAO. However, when forming a train a training set for this problem, we need to know the true location of the sources seen in time-frequency space. Very little, if any, of the collected data has been assigned labels with the true locations of RFI. Thus to create a training set from this data for the DNN to learn, we would first need to search the data and assign the necessary coordinates for the bounding boxes. To circumvent this problem, a synthetic dataset was used.

It is straightforward to create a program that can generate a large number of images, each with predetermined sources in, and provide an associated label to these images. The program used to generate this data is described as follows. A two-dimensional array of fixed-sized is initially filled with noise that is drawn from a random normal distribution. This is to represent the Gaussian thermal noise that would be seen in real observations. To add a source, five random integers are drawn from a uniform distribution to define the length, width, orientation, and the (x,y) coordinates of the lowest left corner. Any entry in the array that would be covered

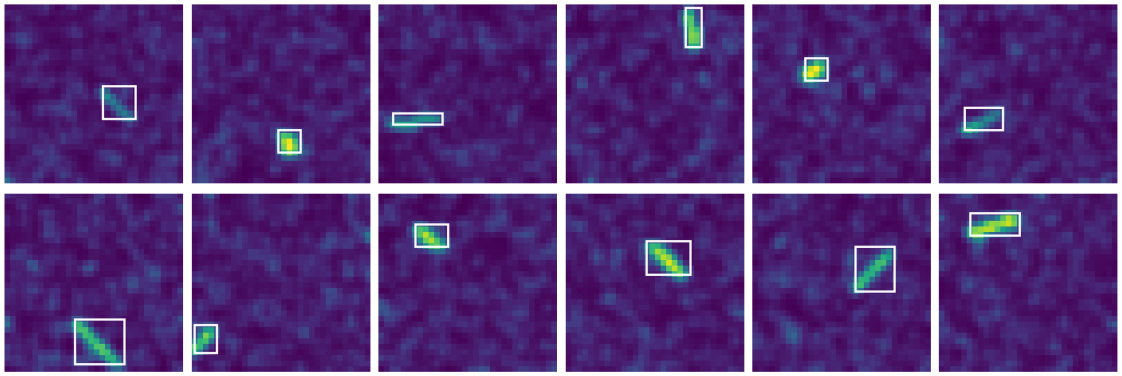

Figure 2.3: Examples of the synthetic data used in training the nerual network

by this source is set to 1. A Gaussian filter is then applied over the entire image and the entries in the array are then clipped to ensure all the values are between 0 and 1. The range for the random distributions and the standard deviation for each distribution are tunable parameters during configuration. Finally, a label is created for each image that defines the lowest left corner and the dimensions of the bounding box. For this initial test the images generated were of size 32 in each dimension, had one source with a length between 4 and 10, width between 1 and 4. The standard deviation of the Gaussian filter was 0.8, and the standard deviation of the noise was 0.2. Figure 2.3 demonstrates the results of using this program to generate sources.

2500 images were generated using the above methodology and split such that 80% went into the train datasets and 20% in the test dataset. The DNN used consists of two fully-connected hidden layers 128 nodes wide with a ReLU used as an activation function. Lastly, dropout was applied to each hidden layer in the DNN. Dropout is a technique used to prevent ANNs from overfitting by randomly disabling nodes in the neural network. By doing so, dropout prevents nodes from becoming highly dependent on each other. For the DNN we used, each node in the hidden layers had a 25% chance of being deactivated during training. The input to the DNN will be the synthetic two-dimensional images and the output will be a vector of length four used

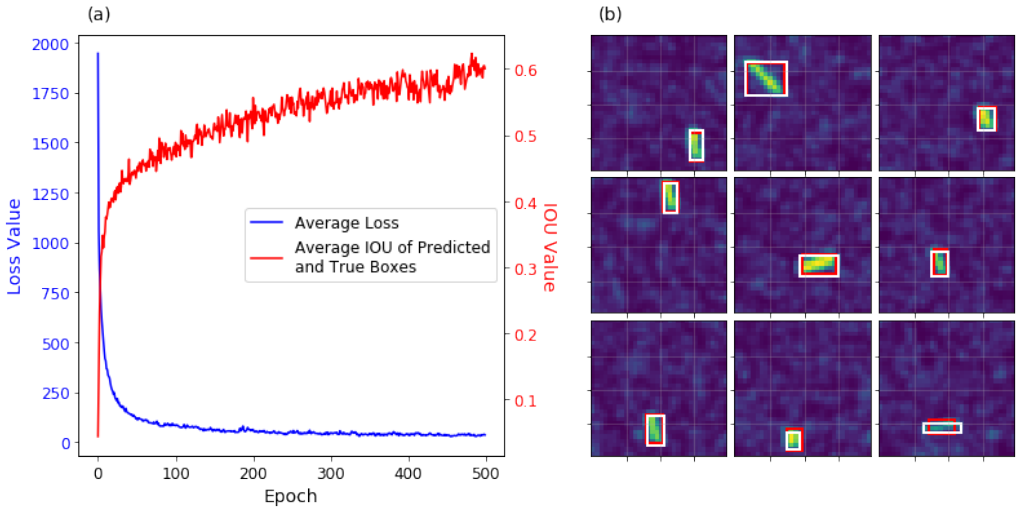

Figure 2.4: (a) The evaluation metrics for training the DNN to predict a single source over 500 epochs (b) A comparison of the predicted bounding boxes, shown in red, and the true bounding boxes shown in white. The predictions are made using the DNN trained for 500 epochs.

to define the predicted bounding box; the two coordinates of the lower-left corner, the width, and the height. The loss function is defined as the mean squared error between the predicted values and the ground truth. In addition, we also track the IOU of the predicted and the true bounding boxes. Figure 2.4 shows how the loss and IOU change during training for 500 epochs.

2.3.2 Multiple Sources

The next step required for this algorithm to used for RFI detection is to apply this process to multiple sources in the same image. To solve this problem we used computer vision methods, in particular methods of object detection, as discussed is Chapter 2.2.1. The YOLO methodology was chosen for this work because an ANN trained on a single tile can be used to search an entire image. Images larger than those in the training set can be searched through by splitting them into the same size tiles as those used for training. This will be important for scanning waterfall plots that cover the

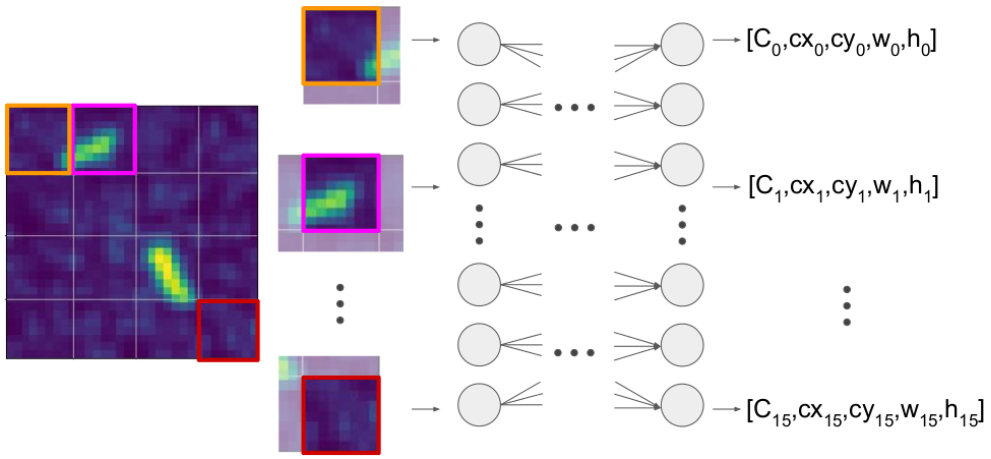

Figure 2.5: The DNN architecture for a 32×32 sized image, split into 16 tiles. Each tile, padded with the surrounding area from the image, is passed into the same network independently to produce 16 output vectors. Each vector contains the confidence score C, and the descriptors for the bounding box.

large frequency ranges required. The downside with YOLO is that if there be more objects in one tile than the network can search for, the algorithm would be unable to detect all of them. In our case, this may work in our advantage. If two sources are so close that the algorithm can not tell them apart, studying them together may be beneficial.

When the image is split up into grid squares, there is information that may be lost on the boundaries. To avoid this, each tile is padded with the surrounding area from the image, in a technique inspired by Extensive Deep Neural Networks (EDNNs) [91]. The key idea is that arbitrarily large systems can be split into a series of non-overlapping focus regions surrounded by an overlapping context region. By doing so, a DNN can efficiently infer extensive parameters such as the number of particles or energy in a system, as shown in the original work. By applying this technique to our images, we ensure that minimal information about a source is lost due to the image being divided. If an image covers more than one tile, there needs to an overlap such that the predicted boxes can be identified as coming from the same

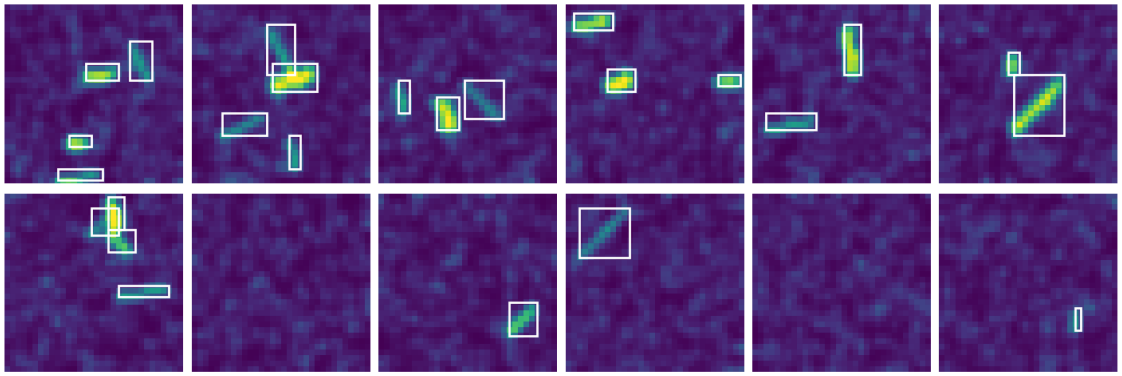

Figure 2.6: Examples of the synthetic data with multiple sources.

source. The neural network architecture that is used with this concept is shown in Figure 2.5.

To test this algorithm, we used the same synthetic dataset as described earlier, but with a random number of sources initialised in the image, shown in Figure 2.6. Each 32×32 image is split into 16 tiles, all with a non-overlapping focus region of size 8. The context region extended each image by 4 pixels in each direction. To train a net on these images, we also need to update the labels associated with them. The label for a single image now become a series of bounding boxes, one for each tile. If there was no source in a particular tile, the confidence and all values for that box were set to zero. On the other hand, if a tile contains the centre point of a source, the label would have a confidence score of one. The rest of the label for that tile is the bounding box coordinates of the source. The ANN used in this test was the same as described previously, but the input now is the size of an individual tile and output predicts the confidence in addition to the box coordinates. Figure 2.7 shows how the the training progressed over 500 epochs.

One important parameter to consider is the size of an individual tile; if it is too large, small objects in the image may be drowned out and missed by the neural net. The tile size should be physically motivated thus some domain knowledge is

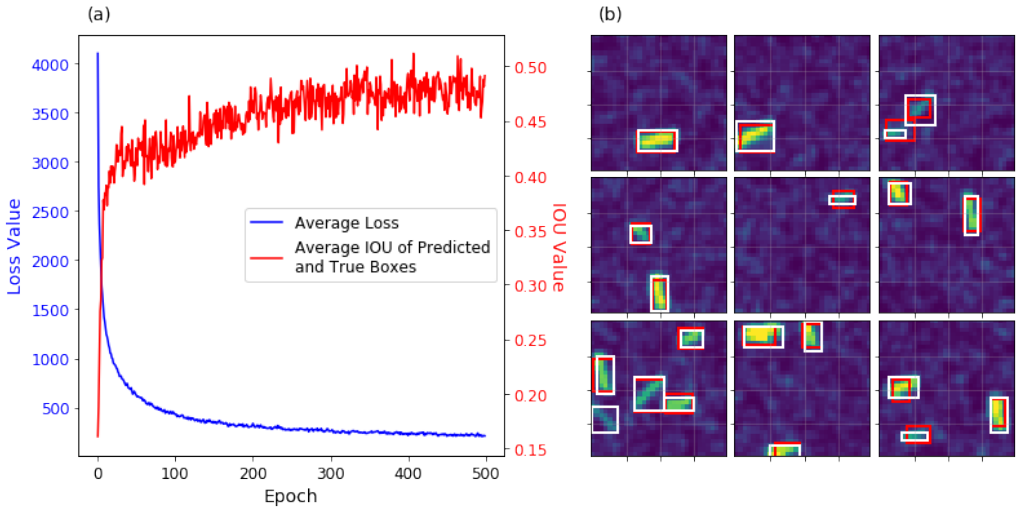

Figure 2.7: (a) The evaluation metrics for training the DNN to predict multiple sources over 500 epochs (b) A comparison of the predicted bounding boxes, shown in red, and the true bounding boxes shown in white. The predictions are made using the DNN trained for 500 epochs.

important. If a single source covers multiple tiles, the algorithm can recognise this and potentially merge multiple bounding boxes.

2.3.3 Applying to Observational Data

The algorithm described in the previous section can obtain the bounding boxes for all sources in a fixed time period. To test for a video or a real-time influx of data, we can simply expand the time dimension. If the total time is longer than what the algorithm can search in one instance, then we can pass a sliding window over to search all of the data. By doing so, we identified the next problem to be solved. The images in the training set are from one instance of time. However, when the window is moved over the data, if the stride is less than the window size, a single source will be detected multiple times. If several sequential frames are passed through the algorithm, then the DNN currently will identify the same object in different frames as a different signal as there is no way for it to tell if it is from the same source or not.

This is especially troublesome if a single event has a higher time duration than what is covered in the window. The program needs to recognise when an object is seen over multiple iterations and identify these detections as one source. Furthermore if a single source pulses then this should also be seen as a single detection.

Therefore, a buffer was introduced to temporarily store recent detections. After every frame, all the detected bounding boxes are passed into this buffer. If there is another source in the same waveband within a fixed time window already in the buffer, then this source and the new detection are merged and the coordinates of the box cover both detections. This means that the two bounding boxes of the same source will be combined when detected in multiple frames. If an object in the buffer has not been updated after a set number of frames, it is removed from the buffer and stored in a more permanent location.

To test how the algorithm performed on a influx of data, we applied the ANN algorithm as described above on data from a real survey taken at DRAO. The survey was ten minutes in duration over a 20MHz range centred at 460MHz. The frequency was split into 20000 channels, each 1kHz wide, and the received power on each channel was recorded every 0.1 seconds. The ANN used was the same network that was created in Section 2.3.2. This means that the network had never seen real RFI data before because it was trained exclusively on the synthetic data, as seen in Figure 2.6. Before the real data was passed into the network, some pre-processing had to be done. This is a necessary step to bring the values of incoming data into the range that the DNN was trained for. Firstly, the bandpass of the filter and background noise had to be removed. To do so, we fitted a Gaussian Process (see section 3.4.2) to the mean received power (in dB) over each frequency channel. We then subtracted the fitted function from the data and clipped the result to be in the range [0, X] where X is some threshold above the noise deviation. Doing so enforces a minimum

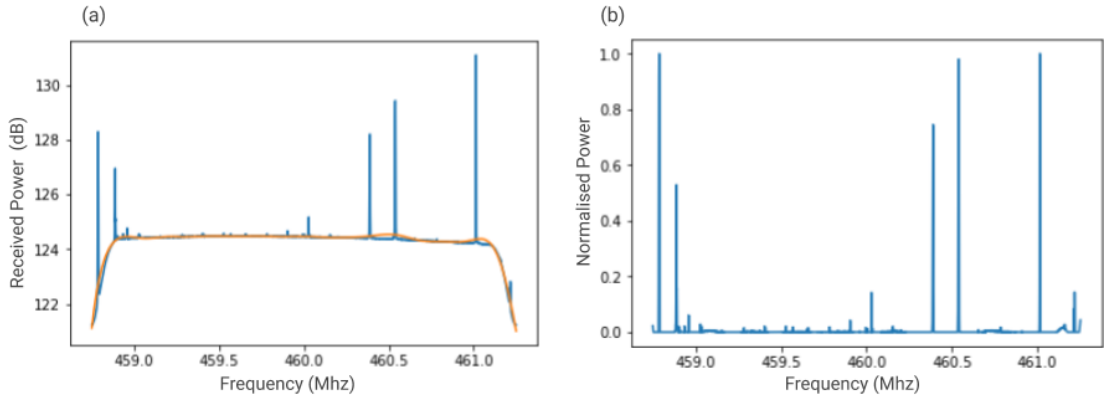

Figure 2.8: (a) The original data mean is shown as the blue line. The Gaussian Process that was fitted to the data is shown in Orange. The fitted function represents the background noise. (b) The data after the background noise has been removed and it has been rescaled.

and maximum value such that the data can be normalised, in this case, we used 5 dB. Finally, we rescaled the values using min-max scaling to bring all values into the range [0,1]. This brings the data inline with the synthetic training set (Figure 2.8).

Figure 2.9 shows a 2.5MHz window for 10 minutes, which gives 2500×6000 data points. The image was tiled up into 32×32 images and passed through the neural network and the detections through the buffer. The predicted bounding boxes are shown by the red boxes that surround the RFI identified. Starting at the lowest frequencies we can see a continuous source detected throughout the entire duration of the survey. In this case, the buffer combined detections if they were within 1.6 seconds of each other; these boxes are so close, most likely overlapping with each other, that they are all merged. Throughout the image, we can see several more transient interference patterns that get picked up as individual detections. Finally, looking at the right-most highlighted area, we can see some very brief pulses followed by a much longer signal. This is an important result as it shows the algorithm can distinguish between different types of events in the same frequency channels.

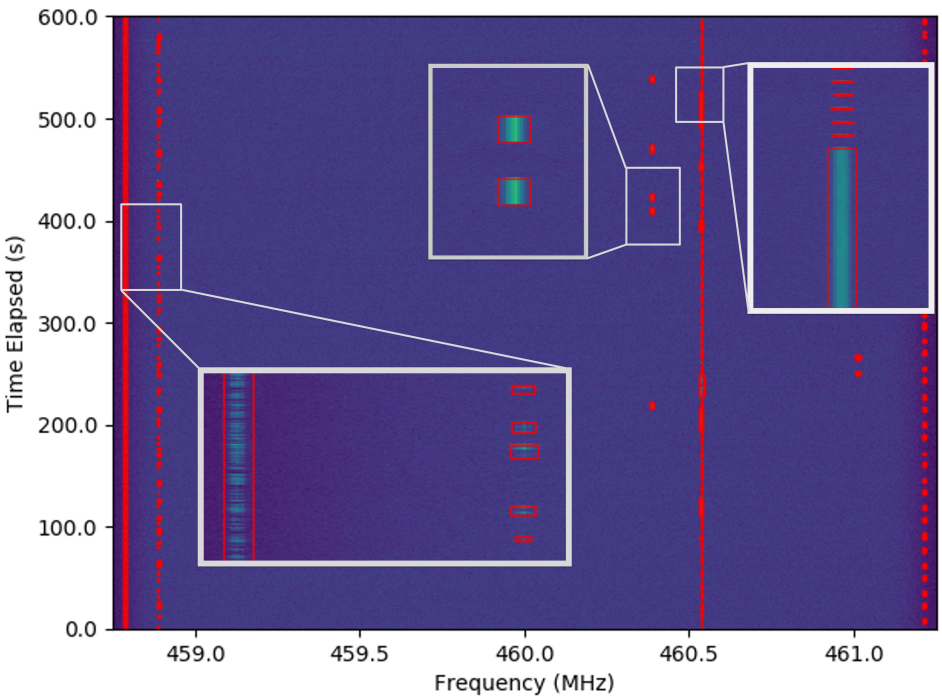

Figure 2.9: 10 minutes of data observed at DRAO over a 2.5 MHz range. The red boxes show the results of the bounding box algorithm. The boxes encompass all of the detected interference across the survey. Three areas have been highlighted to show the results of the algorithm on different signal behaviours.

2.3.4 Further Developments

We are currently still in the testing stage of this detection method; in October the detector was set up by the operations team to run in real-time over a 50MHz bandwidth. There are several aims for this test, the first simply being to see if the monitor can handle the influx of data over an extended period of time. The results produced by the monitor will be analysed and compared against traditional techniques for RFI detection. This will be an important test to see if the ANN monitor can become a standard detection method at DRAO. Finally, the monitor will provide a dataset of bounded transmissions for study. This set will be used as a basis for clustering project that will be discussed in the Chapter 2.3.4. Additionally, by having this data

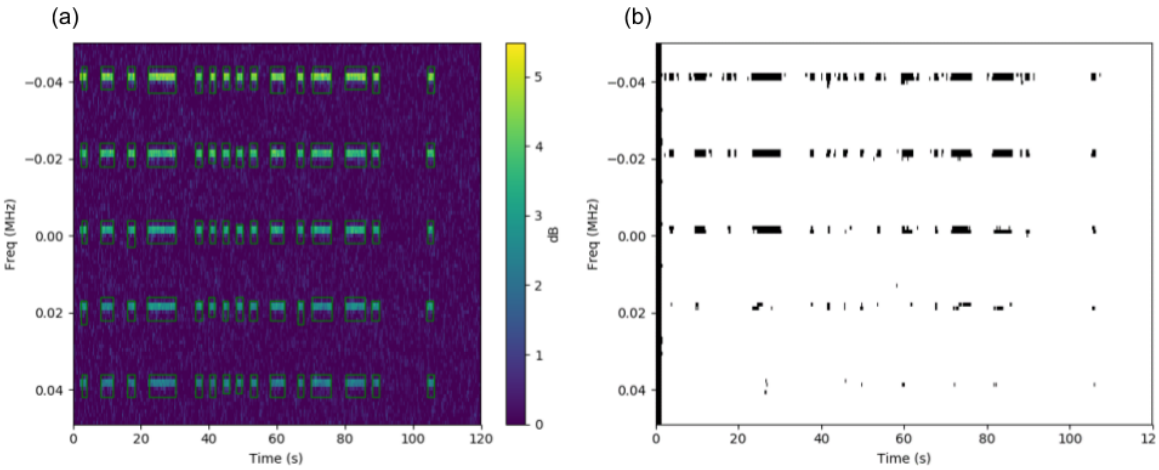

Figure 2.10: A comparison of the bounding box method (left) and the spectral kurtosis test (right). Both are tested on the same image (a) The RFI signals that are detected are shown by the drawn on bounding boxes. (b) The SK test only shows signals that it believes to be RFI, the rest of the signal is removed from the image.

available, it would be possible to retrain the neural networks with a training set that includes real data, in addition to the supplemented data.

While the monitor is still being tested, we can compare it to established and commonly used methods. One such method is SK, as discussed in Section 2.1. To compare the two methods, a test image was generated with a pattern of repeated signals where the signal-to-noise ratio is decreasing by 1dB, starting from 5dB. The bounding boxes obtained from the DNN method are shown in Figure 2.10a and the results obtained using a SK test are shown in Figure 2.10b. The SK isolates segments of the signal that it recognises as being different from the background noise. When the signal to-noise-ratio is high, the bounding boxes match the isolated signal from the SK method, which indicates the two methods have similar performance. However, the DNN method performs much better when that ratio is low. The apparent improvement can be attributed to the fact that the DNN considers the RFI source in the context of the adjacent channels where SK treats each channel independently.

Identifying Novel Signals

Now that we have a tool to detect incoming interference signals, we need a tool to study the RFI being detected. A major goal of this project was to build a rich description of the RFI scene at DRAO so staff could actively police the site on a daily basis. To assist this, we want to create three broad categories for a new identification to fall into. First, there are events which are common but outside of the observatory's control. For example, if a plane flies overhead and produces RFI, it is unreasonable for observatory staff to prevent it so they don't need to be alerted. However, the event will still be logged and the information stored. Second, there are sources that are common but preventable, for example, a cell-phone on site. In this case, a member of staff should be alerted and provided with the appropriate information. The final category of detections is an event with high novelty. A novel event would be identified as a signal which behaves in an unusual manner. The obvious indicator of novelty would be the frequency that the signal was detected at; if a new signal has appeared on a frequency that is typically quiet, that signal would be seen as novel. There are other examples of novel behaviours, such as a change in transmission behaviour, or an always-on signal has turned off. We would like to automatically separate normal events from novel events for further investigation. To detect what is novel in the local radio frequency scene, we need to first characterise what is normal. To do this, we look to UL to understand the structure of the detection data.

2.4 Unsupervised Learning

Consider the problem described in Section 2.2 of classifying the MNIST dataset. With SL, we would train an algorithm to predict the label from the image. The training set example consists of example inputs and their true labels. The algorithm would hope to learn a general mapping of input to the desired output from these examples. However, if the training set did not include labels for the data, the algorithm could not know the desired output. In this case, there is no 'supervisor' to guide the learning of the algorithm, and so a model can not be trained in the same way as in SL. Therefore, the problem is now an unsupervised one.

The goal for any UL algorithm is to identify patterns and trends in the data that are beyond the noise of the dataset [15]. There is no feedback provided to the algorithm, as there was with SL, and instead, it must build its own representation of the input. The method of obtaining this representation varies, often taking inspiration from statistical algorithms, but the goal is standard across models; search for indirect hidden structures of features to analyse the data provided in both the training set and new data. A successful model will be able to group and summarise a dataset in an effective and useful manner. This could mean collect the data into similar groups or compressing the data by reducing the dimensionality.

One of the most common UL problems is clustering. Clustering aims to find structure in a collection of unlabelled data such that individuals in the dataset can be grouped together with others that are similar in some metric. In other words, given a set of points, with some notion of distance between each pair of points, the goal is to obtain clusters of points that have a small distance to other points within the cluster, but large distances to members of other clusters.

Clustering is extremely dependent on the intentions of the researcher; often the process of clustering is seen as one of many steps for obtaining information from the dataset. As a result, there is a large variety of clustering algorithms, each one may work well to suit a subset of needs [92]. One of the more popular clustering algorithms is known as K-means clustering [93]. It aims to split the data into a finite number, K, of discrete clusters. Given a training set of inputs $x_1, x_2, ..., x_N$, the algorithm aims to split the set into K clusters, denoted $\{S_1, S_2, ..., S_K\}$ with means of each cluster $\{\mu_1, \mu_2, ..., \mu_K\}$ such that Equation 2.2 is minimised.

$$m(x) = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2 \tag{2.2}$$

K-means is an iterative process that is initialised by randomly generating K points in the range of the data. These form the first K means. Using these values, each point in the data set is assigned to the cluster with the closest mean. The mean values of each cluster can now be recalculated using an updated set of points in the cluster. The data points are then reassigned, and the process repeats until the clusters no longer change. Figure 2.11 shows an example of clustering using the K-means algorithm.

While being extremely popular, K-means has several issues. As the data is partitioned into K sections, every point in the dataset has to be inside of one cluster, regardless of if it is an obvious outlier. This is to say, K-means provides exactly K partitions in the dataset, not K clusters. This leads to the second problem; the user needs to specify a value for K. This is less of a problem if the dataset structure is known, however in a new dataset, it may be hard to tell what value works best for the problem.

Both of these issues are particularly problematic for our application. We do not know how many clusters we expect to get and our desired use it to identify points

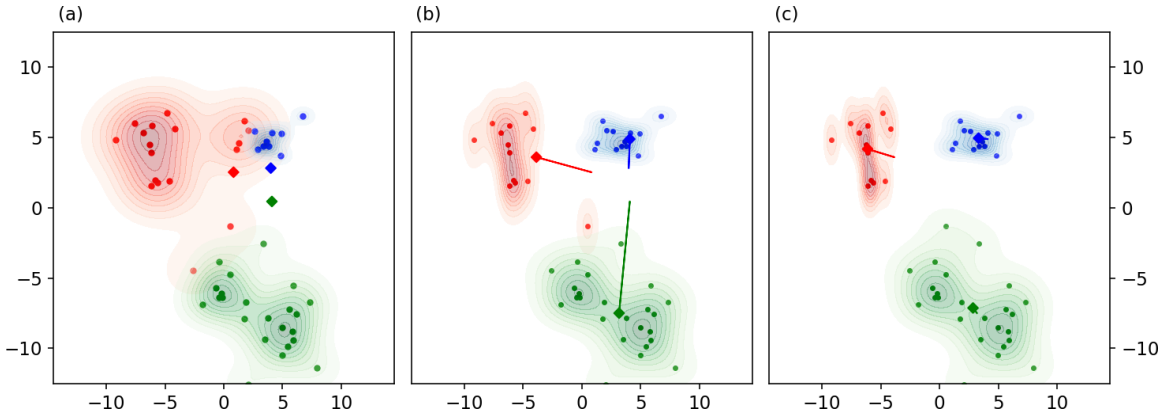

Figure 2.11: Three iterations of the K-means algorithm clustering 50 points into 3 clusters. The means are indicated by the coloured diamonds. (a) Shows the distribution of the data. (b) The mean points are reassigned using the new clusters. (c) The final configuration of the clusters. At this point, the data points can not be reassigned to another cluster.

that do not belong to any known cluster. Therefore we will move away from partition clustering and instead look to use density clustering methods.

2.4.1 Density Clustering Methods

Density-based clustering defines areas of high density relative to the dataset, as clusters [94]. Objects form clusters with other nearby objects, and those in sparse areas of the dataset are considered to be novel data points or noise. One major benefit to this class of methods is that the number of clusters does not have to be known in advance; instead, the number of clusters will be dependent on the density pattern of the data set. How the areas of high-density are defined depends on the algorithm used. Perhaps the most similar density clustering method to K-means is known as Mean-shift. The approach taken by Mean-shift is to move each object towards the densest area in its nearby neighbourhood [95]. By doing so, the algorithm finds all the local maxima of the density function given only the discrete data points. A kernel function K, typically a Gaussian kernel, determines the weight of nearby points used

to calculate the weighted mean of the density as follows.

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)} \tag{2.3}$$

It is an iterative algorithm that sets $x = m(x)$ and repeats the above calculations until $m(x)$ converges. An examples of this algorithm is shown in Figure 2.12.

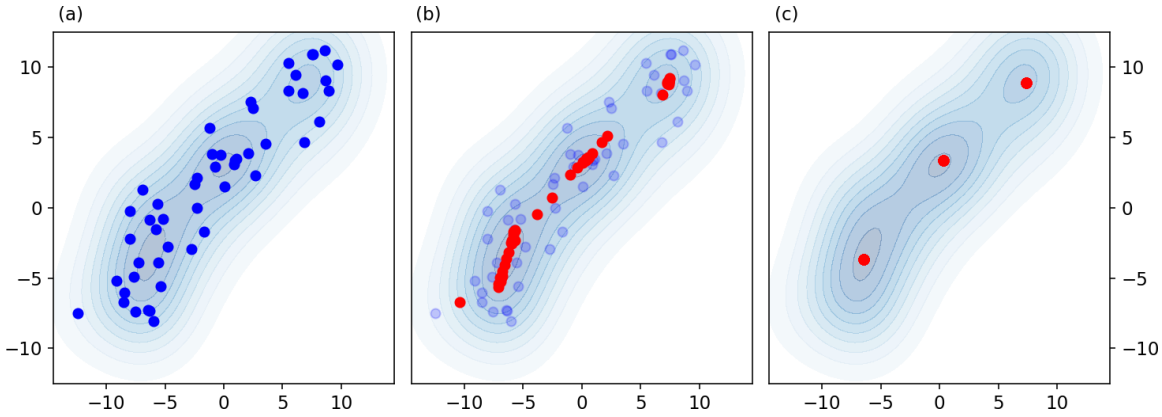

Figure 2.12: Three iterations of the Meanshift Algorithm. The density of the dataset is marked on all three plots. (a) Shows the distribution of the data. 50 data points are used. (b) The first iteration, the red points are the new shifted points. (c) The final set of means found. Clusters will centre on these three points.

While the algorithm achieves accurate estimates of the cluster locations, the selection of the neighbourhood size is not trivial. This parameter determines the how far of each point and an unsuitable window size can cause groups to be merged or small fluctuations in density to form invalid clusters. Another drawback to the Mean-shift algorithm is that it is very computationally expensive due to its iterative nature. While the algorithm is still converging, the points which are settled are still contributing to the number of computations occurring each iteration. As such, we will look to other density-based clustering methods.

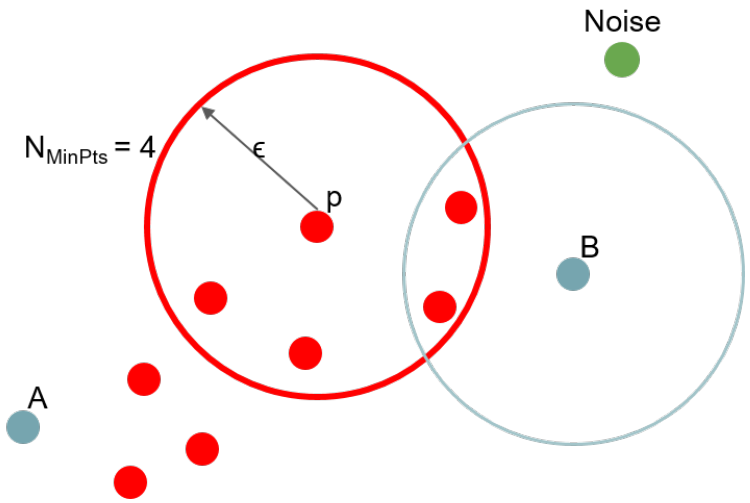

Figure 2.13: Point p, and the other red points, are core points as they have at least N_{MinPts} points in their ϵ-neighbourhood. A and B are both considered border points because they have a core point in their range, but are not core points themselves. A and B are density reachable from each other, as there is a path of connected points. The green point has an empty ϵ-neighbourhood, so it is labelled as noise.

DBSCAN

The effectiveness of clustering algorithms depends on the parameter selection. Often this can require substantial domain knowledge, which may not be known for large datasets. This is one of the problems that the method known as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [96] aims to solve. The algorithm only requires two parameters; the minimum distance between two points for them to be considered neighbours, ϵ, and the minimum number of points to form a dense region, N_{MinPts}. Starting at a random point in the dataset, DBSCAN counts how many data points fall within distance ϵ of that point; these points are known as the ϵ-neighbourhood. If this number is higher than (N_{MinPts}-1), that point is known as a core point and forms the basis of the first cluster. From core point p, any point q that is within distance ϵ is directly reachable from p. These definitions are illustrated in Figure 2.13.

The algorithm then iteratively expands the cluster by going through each point that is directly reachable and searching for additional core points by checking the ϵ-neighbourhood surrounding them. All the core points that are found, plus the additional points within their neighbourhoods, form a cluster. All of these points are density reachable from each other. That is to say, from any one point p, there is a path to any other point q of the form $p, p_1, ..., p_{n-1}, q$, where each consecutive point is directly reachable from the one before. When no more core points are found, the algorithm will then repeat the process starting at an unassigned data point until all entries in the dataset are either in a cluster or labelled as an outlier. An outlier will be labelled as such only if there is no core point within its neighbourhood. This process is outlined in Algorithm 2.1 [97].

Algorithm 2.1 DBSCAN

```
1:  procedure DBSCAN(D, ε, N_MinPts)
2:      Labels = Empty                          ▷ Each point will have a cluster label
3:      count = 0                                          ▷ Current cluster counter
4:      for each point p in the dataset D do
5:          if Labels(p) ≠ undefined then
6:              continue                        ▷ Skip the points already in a cluster
7:          N = Neighbourhood(D, ε, N_MinPts)
8:          if len(N) < N_MinPts then
9:              Labels = Noise
10:         count += 1                                      ▷ Creates a new cluster
11:         Labels(p) = count
12:         Cluster = N
13:         for each point q in Cluster do
14:             if Labels(q) = Noise then
15:                 Labels(q) = C
16:             if Labels(q) ≠ undefined then
17:                 continue
18:             N = Neighbourhood(D, ε, N_MinPts)
19:             Labels(q) = count
20:             if len(N) < N_MinPts then
21:                 continue                    ▷ Only add N if q is a core point
22:             Cluster = Cluster ∪ N
```

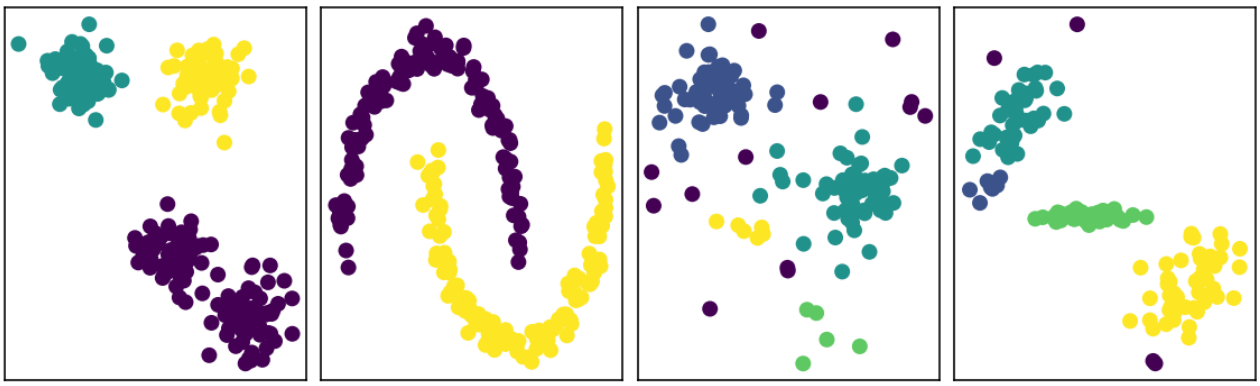

Figure 2.14: The results of the DBSCAN algorithm applied to various test datasets.

The distance metric used when calculating which points are in the neighbourhood is flexible. The euclidean distance between two points is typically used [98] as this is often easiest to understand. However, other metrics can also be used if they fit the dataset, for example the cosine distance [99].

The success and wide use of DBSCAN can be attributed to its robustness to outliers, ability to find clusters of arbitrary shape, and if the data is well understood, effective parameters can be easily set. Figure 2.14 shows how the algorithm clusters various test datasets.

On the other hand, if the dataset is not well understood, choosing a meaningful distance threshold can be difficult. This algorithm also suffers if the density of the data ranges throughout the dataset as the parameters are set globally. There have been many descendants of DBSCAN that aim to solve these problems [100, 101], but we will look at one in particular known as Ordering Points To Identify the Clustering Structure (OPTICS) [102]; an algorithm with the same underlying principles as DBSCAN but aims to solve the problem of obtaining clusters in datasets with large differences in density.

OPTICS

What sets DBSCAN and OPTICS apart is that latter also considers points that are already part of a more densely packed cluster, thus generalising the former. OPTICS creates an ordering of points, first by assigning each point p a core distance, calculated with Equation 2.4 [102]. This is the minimum distance required such that if the ϵ-neighbourhood was recalculated using it, the point p would still be a core point, designated $dist_{MinPts}$. This distance is calculated by taking the closest (N_{MinPts} - 1) points and calculating the distance to the furthest one. For example, if $N_{MinPts} = 4$ then the distance to the third closest neighbour from p is $dist_{MinPts}(p)$. As with DBSCAN the distance metric is typically euclidean but other metrics are viable.

$$\text{Core Distance(p)} = \begin{cases} \text{Undefined} & \text{If } |N(p)| < N_{MinPts} \\ dist_{MinPts}(p) & \text{otherwise} \end{cases} \tag{2.4}$$

Using this distance, we can calculate the reachability distance for a point p with respect to another point q. This represents the smallest distance such that p is directly reachable from q, as long as q is a core point.

$$\text{Reachability Distance(p,q)} = \begin{cases} \text{Undefined} & \text{If}|N(q)| < N_{MinPts} \\ max(\text{Core Dist(q)}, distance(p,q)) & \text{otherwise} \end{cases} \tag{2.5}$$

The OPTICS algorithm calculates the core-distance for all objects in the dataset and the reachability distance from each object to its respective closest core object. Starting a random point p in the dataset, the algorithm will keep a list of all the points within the core distance of p. The ordering will start with the distance to the

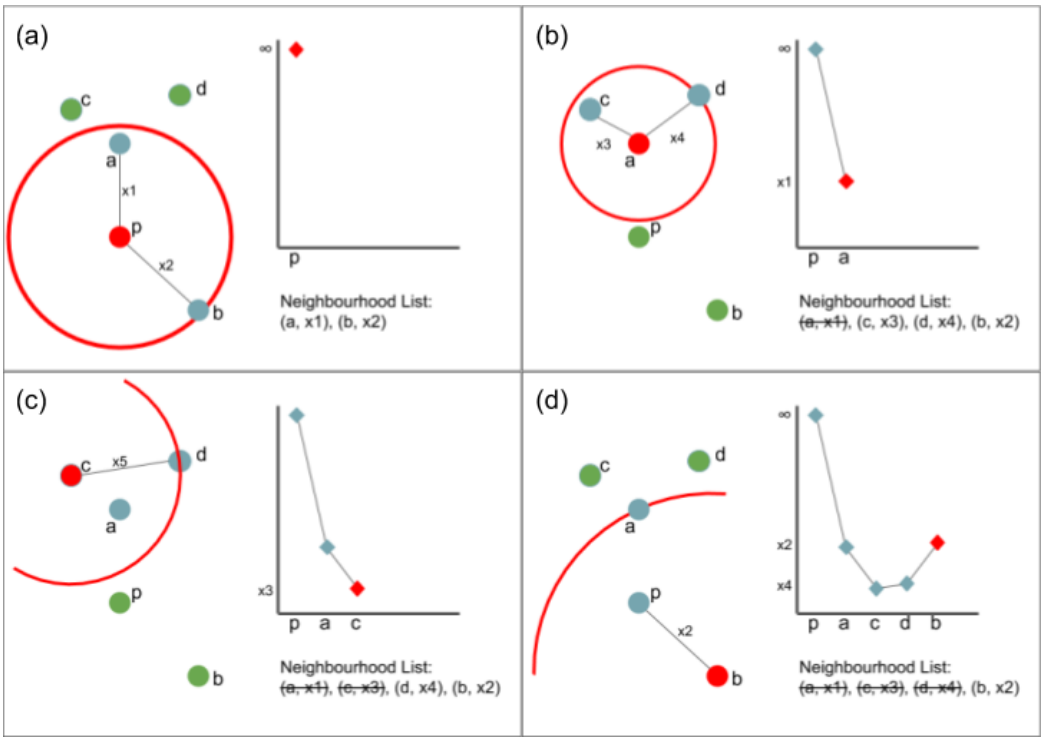

Figure 2.15: For this example, $N_{MinPts} = 3$. The red circle is the core-distance for the current point. (a) Starting with point p, we can see points a and b are within its core-distance. Both of these are added to the list, along with their distances from p. (b) The distance to point a is shorter, and so we add that point to the reachability plot next. It has points c and d within its core distance and so these are added to the list. (c) Point c has the shortest distance from a previous core point, so it is added next. Point d is within it's range, but the distance from c to d is larger than from a to d, so it is not added to the list. (d) c and d have no neighbourhood points not already added, so we go back to b and add that point to the plot.

next closest point, q. From this point, we update the list by removing p and adding in the points within the core distance of q. The next point in the ordering is the point in the list with the shortest distance to either p or q. This process repeats until all the points are ordered. See Figure 2.15 for an illustrated example.

The reachability plot is a valuable visitation tool for allowing the user to see the cluster structure of the data. However, on its own, OPTICS does not segregate the data into clusters. To do so, the reachability plot that is produced needs some form of post-processing. This can be as simple as manually selecting a viable ϵ value,

or series of values, that cluster the data appropriately. Alternatively, this can be automated by detecting the steepness of the curve or searching for local maxima. Hierarchical clusters would need multiple runs of the DBSCAN algorithm but the creator of OPTICS claims that there is only a 1.6-factor slowdown by using OPTICS compared to DBSCAN [102]. Therefore, if little is known about the data, or the density is known to be inconsistent, then OPTICS is often a better choice.

2.5 Clustering Signals

The goal for using these techniques was to characterise the RFI scene around the observatory site using the detections from the RFI monitor. The model created does not have to be static; if we were to keep a history of the points as they are collected, we could regularly update the model such that it is relevant to the recent RFI detections around the site. One case where updating the model is necessary would be after the addition of a new instrument that emits off a radio frequency signal. Immediately after the installation of this instrument, the signal would be novel and thus would send off alerts. However, once this becomes a regular incident, the notification would become unnecessary. In terms of a clustering model, several data points from the same source will stop being anomalous once they form a cluster.

Before we cluster the detections, we first need to explore the dimensions available to us. The detection algorithm in Section 2.3 treats a transmission as an atomic unit by bounding the frequency and time ranges. These bounding boxes use four attributes to describe the signal; time of detection, duration, centre frequency, and bandwidth. The bounding boxes are also used to extract short segments from the original waveform of the signal, giving us access to much more information about the signal. For example, we can further describe the event using statistical parameters of

the signal such as the higher-order cumulants $C_{4,2}, C_{6,3}$ (Equation 2.6) [103]

$$C_{4,2} = E(|x|)^4 - |E(x^2)|^2 - 2E^2(|x|^2) \tag{2.6a}$$

$$C_{6,3} = E(|x|^6) - 9E(|x|^4)E(|x|^2) + 12|E(x^2)|^2 E(|x|^2) + 12E^3(|x|^2) \tag{2.6b}$$

The cumulants in particular are valuable parameters in signal identification as they have been used in the past to distinguish between different modulation types. [103,104]

We aim to use features extracted from the original signal, in addition to those provided by the bounding boxes, to cluster the data into behavioural groups. We hypothesise that as RFI samples are collected, clusters will start to form in behaviour space and groups of typical/common sources such as cell phones and cars on the nearby road will be identifiable. There are also some other patterns that we expect to see emerge from the model. For example, RFI activity will likely be higher during working hours when more people are on site because of their personal devices that they bring with them.

To begin with, we clustered the data in two dimensions. This allowed us to observe the clusters formed in an easy-to-understand manner and to ensure the OPTICS implementation, as well as the cluster extraction, are both working as required. To identify the clusters, we used be using a gradient clustering methodology [105]. In a reachability plot, clusters are represented by valleys in the data. Thus the key idea is to look for large gradients in the reachability plot that indicate either the beginning or the end of a cluster. A cluster will consist of a steep downward area, followed by at least N_{MinPts} number of points, and finally a steep upward area. Some post-processing was done on the clusters to remove those that are too large, beyond 50% of the entire dataset, and also merge pairs of clusters that share more than 75% of the same points.

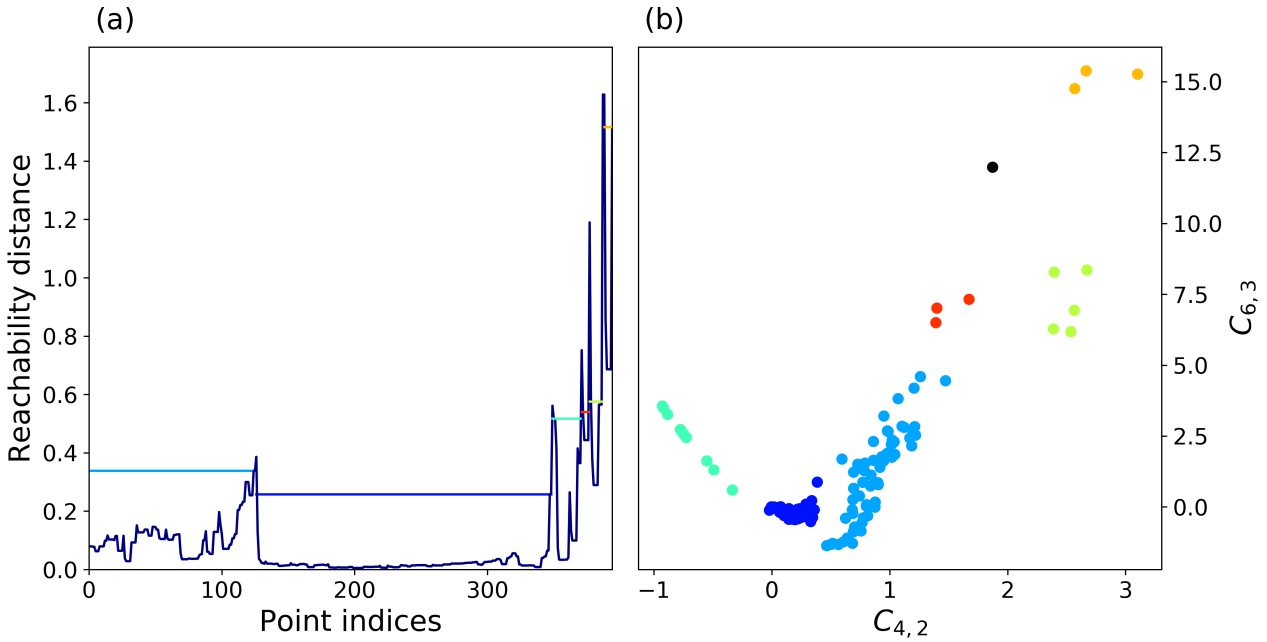

Figure 2.16: (a) Shows the reachability plot for the cumulant data. The detected clusters are shown by horizontal coloured lines over the respective index range. (b) The $C_{4,2}$ and $C_{6,3}$ values are plotted and with the identified clusters shown with unique colours, matching those in (a).

The two dimensions we used were the higher order cumulants of all the signals detected in the 10 minute test survey that was shown in Figure 2.9. We expected several clusters to emerge to that would represent different devices causing the interference. Additionally, when a signal has a low signal to noise ratio, the values of cumulants tend to zero, therefore we also expected a cluster centred at (0,0) for these kind of signals. Figure 2.16a shows the reachability plot as output from OPTICS. The predicted clusters obtained using gradient clustering are drawn on the reachability plot and these clusters are shown in Figure 2.16b. The three largest clusters are approximately between 0-120, 120-350, and 350-370 corresponding to the clusters between -1.0 and 1.5 in the $C_{4,2}$ dimension. As a result, we only now need to manually create three labels that will cover nearly 90% of the dataset. For example, the clus-

ter with indices approximately at 350-370 was identified as FM voice transmission devices used by emergency services. This means if we were to receive a new data point that was positioned in the same area as the original FM transmissions, it could automatically be assigned that label.

To explore the data, we will use parallel coordinate plots [106]. These plots are comprised of n parallel axis, one for representing each dimension of the data. The premise is that a single data point in n-dimensional space is visualised as a single line with one vertex on each of the parallel axes. Visually plotting the data in n-dimensions provides extra insight into the detections that may not have been easy to identify beforehand, especially when providing a visual aid for others not familiar with the project. Figure 2.17 shows a parallel coordinate plot for six dimensions of the data, represented by the six parallel vertical axes. By colouring the data lines in adherence to the colours representing the clusters identified by the OPTICS algorithm, we can get extra insight from the clusters. For example, we can see that all points higher than 52 dB belong to one cluster. This is the same cluster that was identified for use by the emergency services. The FM transmissions need to be powerful to well received across large distance and so the received power that is observed aligns with this purpose.

While the information gained previously from the two-dimensional clusters is valuable, it is not a complete representation of the dataset. Because we have only used the higher-order cumulants, which represent the modulation parameters, the predicted clusters could, at most, give information about the source device. However, there are additional patterns which can be useful, for example, the temporal patterns of the interference received. If we were to find patterns in the temporal structure of the data, over the span of a day or week for example, then this could help identify staff habits that lead to interference.

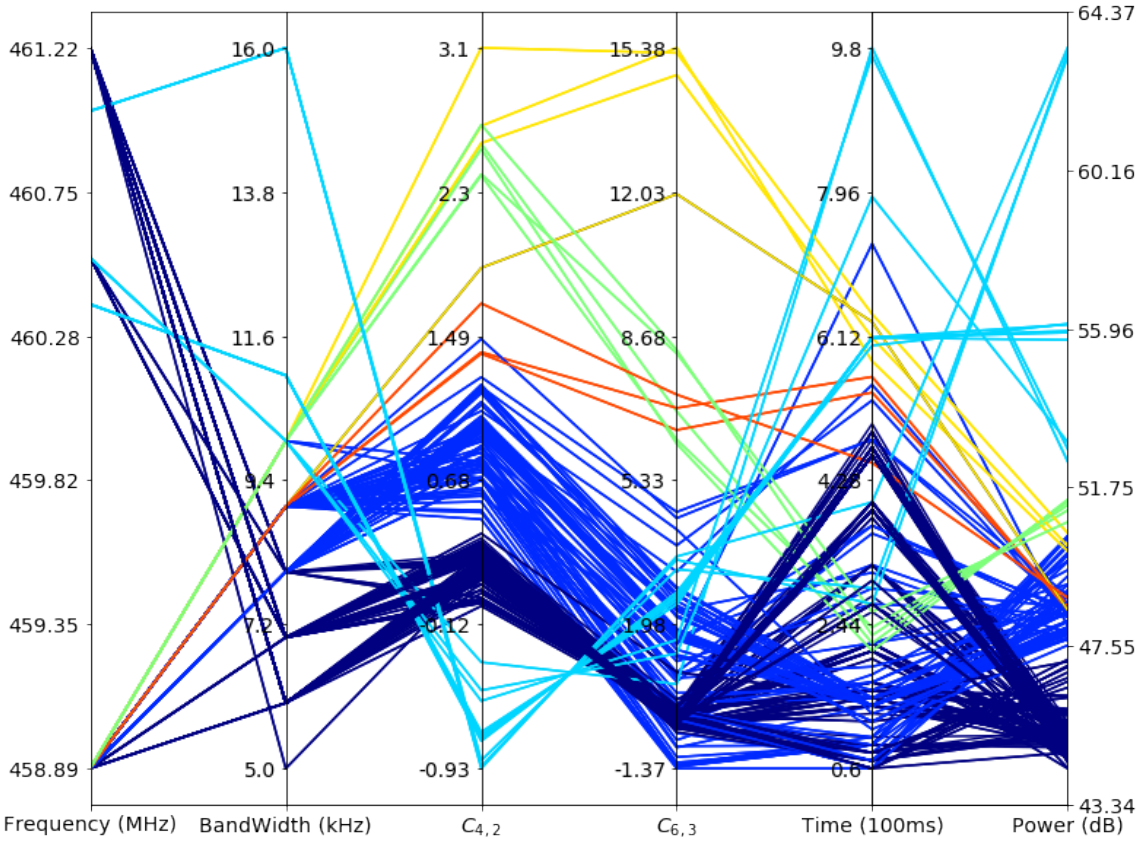

Figure 2.17: The same points shown in Figure 2.16 are now visualised using a parallel co-ordinate plot. Each six-dimensional point is a single line the connects each of the parallel axes. The clusters that were identified using gradient clustering have unique colours to identify them.

This is all to say that expanding the clustering effort to all dimensions of the data is going to be extremely valuable moving forward. To truly gain as much information as possible from the dataset, we would ideally cluster the data in several subsets of the available dimensions. As we have shown, clustering in just a subset of parameters can provide valuable insights, but this may not always be the case. Therefore, in the future, we will be looking at methods of clustering in higher dimensions and identifying the dimensional subsets of value to the observatory staff.

2.5.1 Further Developments

The problem of effectively identifying novel sources is still yet to be completely solved. Up until this point, we have only clustered the data using two dimensions at a time. However, the data collected and results from the immediate analysis provide plenty of valuable parameters. In this section, we will introduce some of the alternate approaches that are currently in development.

Adding Expert Knowledge

When we discussed the density clustering methods DBSCAN and OPTICS in Section 2.4, one of the proposed benefits to using these methods was that limited domain knowledge was required by the user. However, the operations team at DRAO have been working around RFI signals for decades and therefore have plenty of domain knowledge. For example, we know that there should be no signals detected in the protected bands, so any signals found there should be treated with high novelty regardless of any other characteristics. This knowledge can be incorporated into the clustering algorithms used. To do so, if a detection has a frequency bounded by the protected bands, then we can ensure that this point is an outlier by increasing the distance from the detected point to all others. If the distance is made to be effectively infinite from these detections to any other, we will always identify them as novel signals. This rule, along with others, are currently being tested using the DBSCAN algorithm for clustering.

Subspace Clustering

It is clear that when discussing the novelty of a signal, we need to consider how imapce each signal parameter has on the novelty. As we have discussed, frequency

is a big indicator of novelty. On the other hand, there are some parameters that, depending on the source, have little impact on if the event is a novel one. As a result, we may see certain subsets of the data cluster well in different subspaces of the parameter space. If this is the case, a global clustering method will struggle. Subspace clustering methods have been developed to solve this problem; one such method is known as subspace PREference weighted DEnsity CONnected clustering (PreDeCon) [107].

When using PreDeCon, each point is assigned a subspace preference. A point has a high preference to a dimension if there is an available neighbourhood within a small distance in that dimension. PreDeCon builds upon DBSCAN and uses the same notion of density connected clusters. The difference with PreDeCon is that the distance metric is adjusted such that the dimensions with high preference are weighted more. We are currently running tests to see how adding preferences into the distance metric alters the clusters that are formed.

Autoencoders

Lastly, there is an alternative approach currently being worked on that makes use of autoencoders as a method of anomaly detection. An autoencoder is a form of ANN architecture that learns to output a copy of the input, through a series of hidden layers [108]. The aim of doing so is to learn a representation of the training set by reducing the width of the hidden layers, creating a lower-dimensional space that the data needs to pass through. The autoencoder can therefore be split into two parts; first, the data is encoded into the smaller latent space before then decoding the data back to the original space. Figure 2.18 shows an example of the architecture for an autoencoder. The network learns to reduce the reconstruction error and thus will learn the most important attributes form the training set and how to use these to

reconstruct the original data.

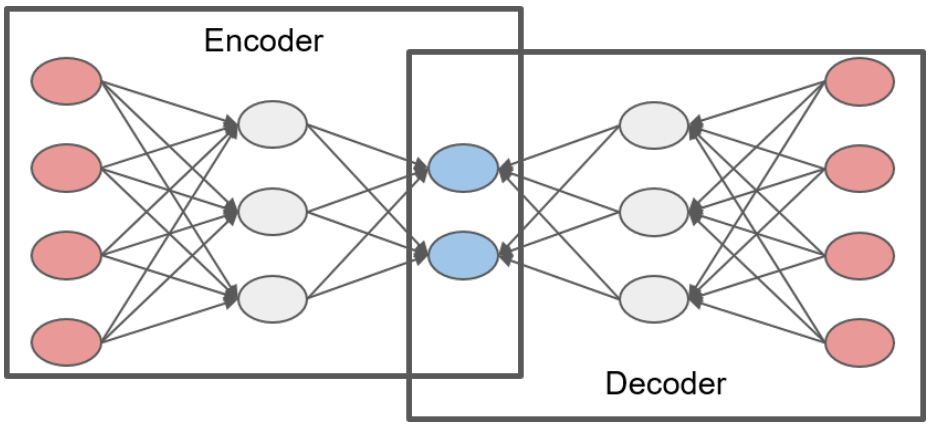

Figure 2.18: An example of a simple autoencoder architecture. The encoder forces the data to a lower space representation, the decoder aims to reconstruct the data with as little distortion to the original image as possible

An autoencoder is being developed to learn how to reconstruct detected RFI signals. The concept is that an autoencoder would be trained with commonly observed signals, i.e. signals which are not novel. When new signals are detected from the monitor, we would then pass them through the autoencoder. If the autoencoder can successfully recreate the signal, then it is similar to those in the training set. However, if the signal is not commonly seen and therefore not in the training set, then the autoencoder will perform poorly when recreating it. Therefore we can use the reconstruction error as a way to tell if the signal is anomalous.

3 Reinforcement Learning and the Scientific Method

The third branch of ML that we will discuss in this work is RL. RL is the study of learning how to maximise some reward signal over many steps by mapping situations to actions [16]. The field encompasses goal-oriented learning and decision making of the learners, also known as agents. A classical formalisation of this problem class are presented as Markov Decision Processes (MDPs), which provide a framework for learning through interaction to achieve a goal. An agent typically starts with no knowledge of the environment it is in, the decisions it can make, or how to obtain its goal. For any given timestep t, the agent must choose an action to take, A_t, given its current situation in the environment, also known as the state, S_t. Beginning with no knowledge of the situations or actions, the agent is rewarded or punished for taking the correct or incorrect actions, respectively. After each action, the agent will get some scalar reward value R_t; the value of the reward will be higher if the action taken was beneficial to the agent than if it was detrimental. Using this information, the agent must learn to map any given state in the environment to the best action available to it, according to previous experiences of the reward signal. The agent-environment interaction is visualised in Figure 3.1.

For a problem to be considered an MDP, all states in the environment must include

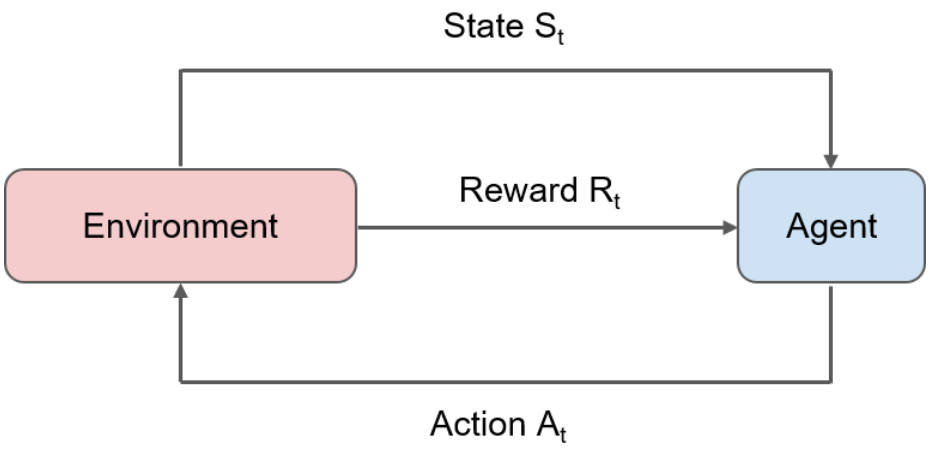

Figure 3.1: Simplified illustration of the standard reinforcement learning dynamic.

enough information such that the agent does not need any additional information to select an action. In other words, an agent only needs the current state to make an action, which is known as the Markov Property. An MDP is said to be finite if the sets containing the states, actions, and rewards are all finite in size. These sets are denoted as $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$, respectively. In this case, a finite MDP can be completely defined by the following function

$$P_a(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{3.1}$$

for all $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$. Due to the Markov property, S_t and R_t only depend on S_{t-1} and A_{t-1} and so P_a defines the dynamics of the environment completely.

RL differs from SL because there are no external true labels presented to the RL learning algorithm. With SL, the correct labels are provided, which allows for the algorithm to evaluate the accuracy of the prediction. RL agents are not provided with the correct action to take, either before or after they take it. Instead, they must learn from their own experience. While UL algorithms are also not provided with the

ground truth during training, they are still classified differently than RL. UL aims to find structure in unlabelled datasets, whereas RL is maximising the output from a reward signal over a single trajectory, which is known as the return. The return is typically a sum of the discounted reward values that were returned over the entire trajectory.

One of the key characteristics that define RL is the compromise between taking actions the agent knows to be good, known as exploitation, and taking actions the agent has little information about, known as exploration [109]. The agent must exploit actions known to be successful to increase the return from the reward function. However, it must also explore new states to possibly find more productive routes to the goal. The agent can not only choose to explore or exploit and be successful; a balance must be found between the two. If there is stochasticity in the system, then the agent must explore each action multiple times from the same state to obtain a true estimate of the value of that action in that state.

Since the early days of RL, playing games has been seen as a natural fit. Success playing board games such as chess or Go [110, 111], as well as video games like Breakout and Pong [112], show the viability of RL in this aspect. Games of all types typically have some form of game board, a limited number of actions the player can take, as well as a score that indicates the player's performance. To see how a game translates to RL, let's look closer at Pong, the two-dimensional Atari table tennis game [113]. To interact with the game, the player gets all of their information from the screen. The current frame shown on the screen is the current state of the game. The pixels represent where both the panels and the ball are located in the current timestep. Based on this information, the player can do one of three things; It can either move their paddle up, down, or not move at all. These are all of the actions available to the agent. To play the game successfully, an agent must move the panel

to hit the ball and hopefully score a point; the score of the game is the reward signal provided to the agent. Therefore the agent will see the screen, move the paddle accordingly and try to win the game by scoring points.

The goal of any RL algorithm is to learn the best policy for the environment [16]. The policy determines the behaviour of the agent by describing the strategy that the agent will employ in all states of the environment to decide the next action. Therefore, when the agent selects an action that is followed by a low reward, the policy may need to be updated to increase the reward. Deep-RL makes use of DNNs to act as the policy. The state of the environment is input to the neural network, and it outputs an action for the agent to take. For a discrete action space, the network would have an output node for every action available to the agent. To pick an action, the agent will typically choose the action with the highest value.

RL algorithms can be roughly split into two categories, model-free and model-based [114]. Simply put, model-based aims to understand the environment and create a representative model such that the algorithm can plan the optimal trajectory. On the other hand, model-free RL learns the optimal policy directly, such that it can choose the best action for any given state. In the classic game examples listed earlier, model-free RL was used. The agent could see the game state and choose an action based on this. Both methods have their shortcomings; Model-Free RL is typically very sample-inefficient, relying entirely on trial-and-error methodology. Sample-inefficiency is not an issue in environments where sampling is cheap, for example, if the environment is fully simulated, such as video games. However, for cases where this is impractical or costly, such as robotics, a model can be created that the agent can use to make decisions. While this does drastically reduce the number of samples needed, the best possible performance is limited by the accuracy of the model. Therefore, model-free algorithms typically achieve better final performance [115].

An example of a successful model-free RL algorithm is known as Asynchronous Advantage Actor-Critic (A3C) [116]. As an actor-critic method, A3C uses a single DNN to learn both the optimal policy and value function. The neural network takes in the state as input but makes two predictions by splitting into two sets of independent fully-connected layers. The portion of the network that estimates the policy is known as the actor and dictates the actions that the agent would take. On the other hand, the portion of the network that predicts the value function is known as the critic. The value function is an estimate of how good a certain state is to be in and the A3C algorithm uses it to intelligently update the policy.

A typical policy gradient implementation uses the discounted reward to update the policy. By doing so, the network is encouraged to take actions that yield a higher reward. However, A3C improves upon this method by calculate the advantage of the current state-action pair. The advantage is used to determine not only how good the action was for the specific state, but also how the result from taking that action in that state differs from what was predicted. A3C used the predicted value function to estimate how good the state is. By using the advantage to update the network, the algorithm will be forced to focus on areas where it's prediction on the value function is inaccurate. In the A3C method, the advantage is typically calculated as follows:

$$A(s) = r + \gamma V(s') - V(s) \tag{3.2}$$

Finally, A3C is said to be asynchronous due to the use of multiple agents to gain experience faster than a single agent could. In A3C a single global network spawns multiple worker agents, each originally a copy of the global network, and each of these agents interact with an independent copy of the environment. The agents all act simultaneously and will ideally gain diverse experience in the environment. The

worker agents will use this experience to calculate the gradients of the loss functions for both the value loss and the policy loss. The workers will then use these gradients to update the global network, and the process repeats. The loss functions used to calculate the gradients are given as follows:

$$\begin{aligned} \text{Value Loss} &= (R - V(s))^2 \\ \text{Policy Loss} &= -log(\pi(s))A(s) - \beta H(\pi) \end{aligned} \tag{3.3}$$

where $H(\pi)$ is an entropy term that encourages exploration by increasing the spread of action probabilities, and β is a scaling term for the entropy value. Using the loss functions, the weights are updated for the respective portion of the network. The original authors of the A3C method state that by having one DNN predict both the policy and value function gives more stability and improves training time. We will be using the A3C method as a comparison for methods presented in Section 3.2.

3.1 Benchmark Environments

As with other branches of ML, RL has greatly benefited from having universal environments for researchers to test their algorithms. OpenAI Gym [117], for example, is an open-source collection of environments designed to be easy to use, have a variety of challenges, and provide standardisation for researchers to test new algorithms. Some of the environments that are included in OpemAi Gym are classic control problems from RL literature, classic Atari games from the Arcade Learning Environment [113], and robot simulations that use the MuJoCo [118] physics engine. Since the release, some of the most impactful papers in the field of RL have used the OpenAI Gym [112, 116, 119–121].

As many algorithms are now being tested in the same environments, direct com-

parisons between them are easy to make. Such et al. (2017) [122] compares the results of using traditional RL methods, Q-learning and policy gradients, to evolutionary learning, which is discussed in Chapter 3.2. The authors also compared the algorithms against a random search of the policy space. The algorithms are tested on thirteen Atari games with each of them performing best on three, except for the policy gradient method which has the best performance on four. The random search method does not perform best in any of the environments. The algorithms all excelled at different challenges and were inferior in others. This shows the importance of fair testing on standardised environments. We have used the classic control environments to test the algorithms presented in Chapters 3.2 and 3.3.

The first classic control environment is the Cart-Pole task [123], also known as the inverted pendulum task. The environment consists of a cart that can move vertically along a frictionless track with a pole attached by a joint to the centre of the cart. The goal for the agent acting in this environment is to keep the pole upright only by moving the cart. An agent acting in the environment has access to the current position of the cart, the velocity of the cart, the angle of the pole, and the velocity of the pole at the tip. The actions available to the agent are to move the cart by applying a horizontal force to the cart in either the $+x$ or $-x$ direction. The game ends when the pole has either fallen over (Fig. 3.2b), the cart has moved too far from the centre (Fig. 3.2c), or the maximum number of steps has been reached.

The second classic control environment is the Mountain-Car problem [124]. This environment tasks the agent with driving a car up a steep hill (Figure 3.3). The car does not have the power required to accelerate up the hill, and so it must move away from the hill first to build up momentum (Figure 3.3b). The agent has access to the current position and velocity of the car. Using this information, the agent must choose to accelerate left, right, or in neither direction. The reward signal is -1 for

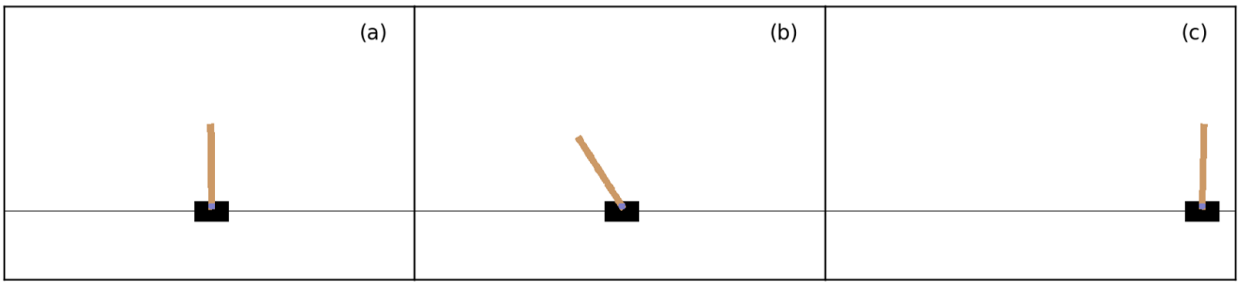

Figure 3.2: The Cart-Pole environment. (a) Shows the initial situation of the environment. (b) Shows a failed state of the environment due to angle of the pole. (c) Another failed state, this time because of the distance the cart is away from the start.

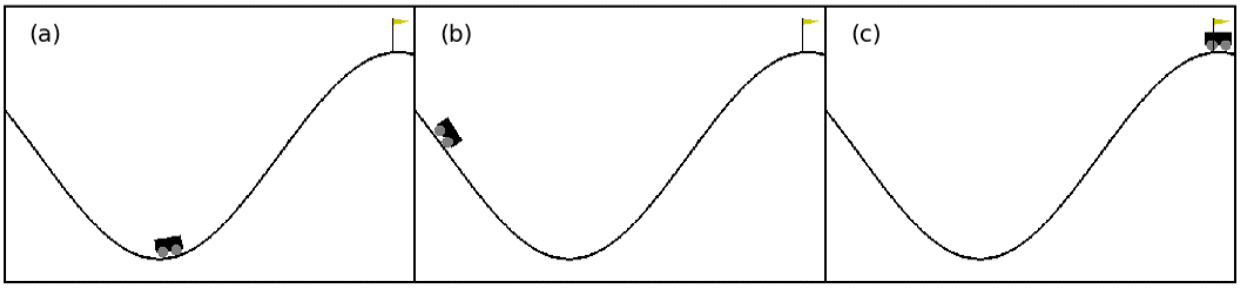

Figure 3.3: The Mountain-Car environment. (a) Shows the initial situation of the environment. (b) The car must move left up the hill to gain potential. (c) The game ends when the car makes it to the flag.

each timestep, and the game ends once the car has reached the top of the right hill, or the maximum number of steps has been reached. The reward scheme means the agent does not receive any information until it has already reached the goal (Figure 3.3c). Regardless of how far the car gets up the hill, if the agent reaches the maximum number of steps before completing the game, it will still get the worse possible score. Thus the agent must learn to get up the hill with no indication from the reward signal.

Finally, the last environment consists of a double pendulum, known as Acrobot [125]. The system has two joints and two links that are initially hanging downwards. The links are connected by an actuated joint, and one of the links is connected to a

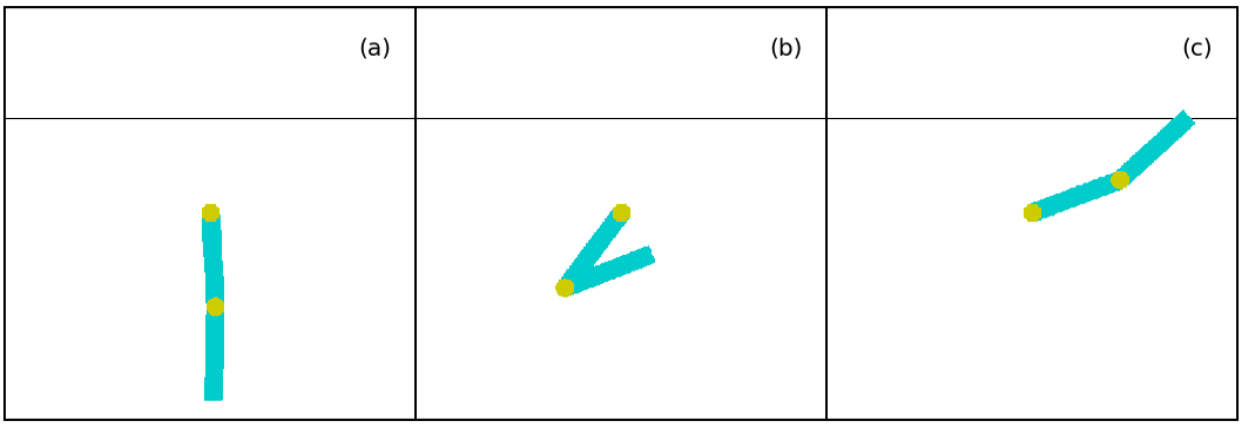

Figure 3.4: The Acrobot environment. (a) Initial situation of the environment. (b) The agent needs to swing the pendulum. (c) Game ends when the tip of the lower link reaches the target line.

joint that is fixed in space (Figure 3.4). The goal of the problem is to get the bottom link up to a target height. To do so, the agent must apply a torque to the actuated joint. The torque can either be positive with a fixed magnitude, negative with the same magnitude or have zero magnitude. The agent receives information about the angle of both the links and their velocities.

We will be using these environments to test algorithms in Chapters 3.2 and 3.3. To provide a comparison, the results from the A3C algorithm are shown in figure 3.5. We can see that the algorithm took by far the longest to achieve a solution on the Mountain-Car task. This is due to the spares reward scheme that is seen in that environment. The agent needs to fully achieve the goal to get a different reward. Once it has a score above the minimum, the A3C algorithm quickly improves the performance. However, there is no reward signal before then, and so the algorithm struggles to improve.

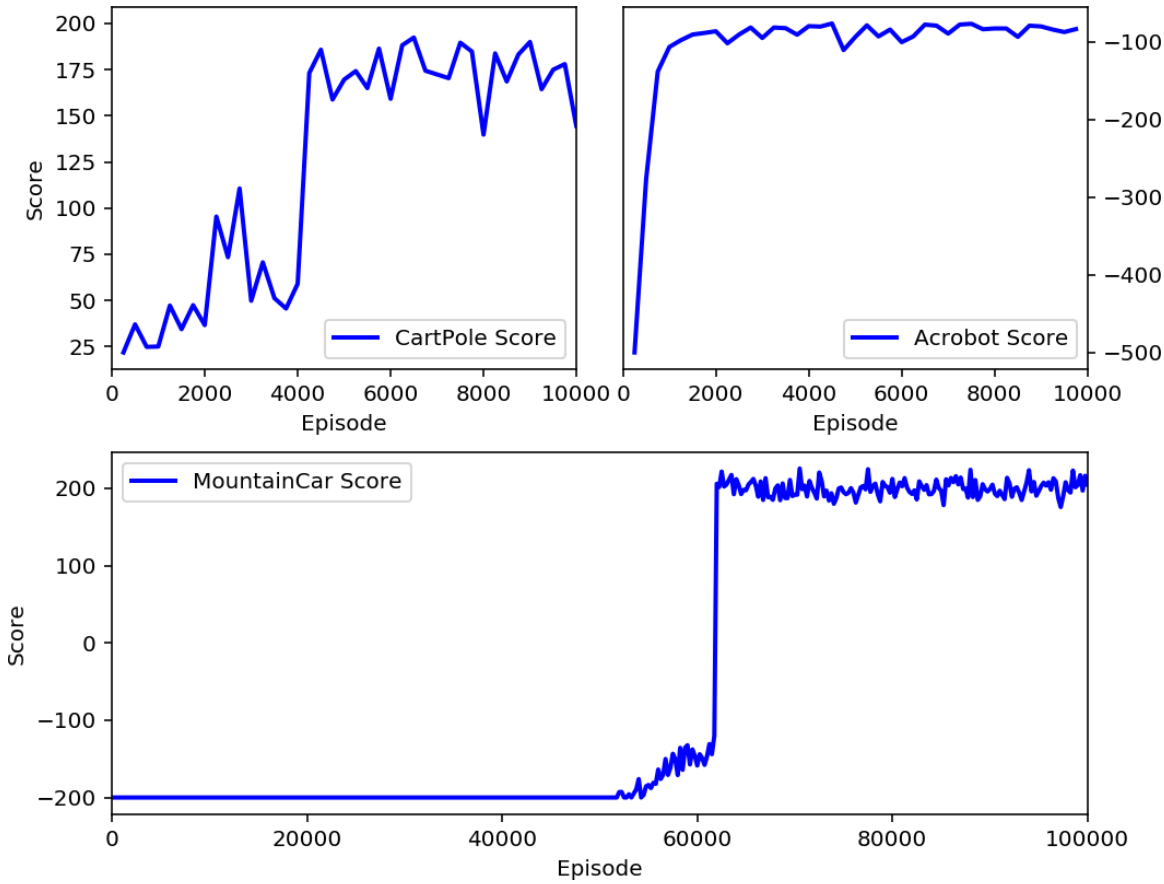

Figure 3.5: Results from the A3C algorithm ran on the classic control environments.

3.2 Evolutionary Algorithms

One approach to learning the optimal policy is to use Evolutionary Algorithms (EAs). As the name may suggest, these algorithms are inspired by biological and natural principles. Any algorithm that can perform optimisation and has the capability to evolve is known as an EA [126]. EAs maintain a set of solutions, known as a population, where each solution is referred to as an individual. The population will maintain a fixed number of individuals, given by the variable n_{pop}. Every individual has a fitness value associated with it that is assigned using some performance evaluation. An EA will then give preference to members of the population that have higher fitness when

creating the next set of solutions.

Let us consider the problem illustrated in Figure 3.6. In this example, we are trying to obtain the largest $f(x)$ value from Equation 3.4. A valid solution to this problem is any x value that is within the acceptable range [-2,6]. Therefore, the population is a set of single numerical values. To initialise the population, we will need to generate n_{pop} individuals. The simplest way to do so is to draw from a random uniform distribution over the solution space. In most cases, the solution will not be obvious and a random distribution will spread the individuals out. To further ensure an initial spread of solutions, the domain can be split into a grid, where each grid cell contains a population member. Note that there are ways to alter the initial conditions to reflect domain knowledge. For example, using an alternate distribution such as a Gaussian, or if splitting the solution space into a grid then having a variable grid cell size allows the density of solutions to be customised.

$$f(x) = x^2 - 10cos(2\pi x) \tag{3.4}$$

For this example, we have created a population of size 10 drawn from a random uniform distribution in the range [-2, 6], illustrated by the red markers in Figure 3.6. The fitness value of any solution, x, is given by the $f(x)$ value. Many of the solutions in this initial population do not perform well; several are very close to a local minimum. However, there are two in particular which stand out from the rest on the right-most side of the figure. These two population members both have fitness functions higher than 20, which is approximately half the optimal value.

With the first population initialised, we now need to select the best candidates and begin to improve the solutions. During the optimisation process, individuals will undergo several operations that change the solution, allowing the algorithm to explore

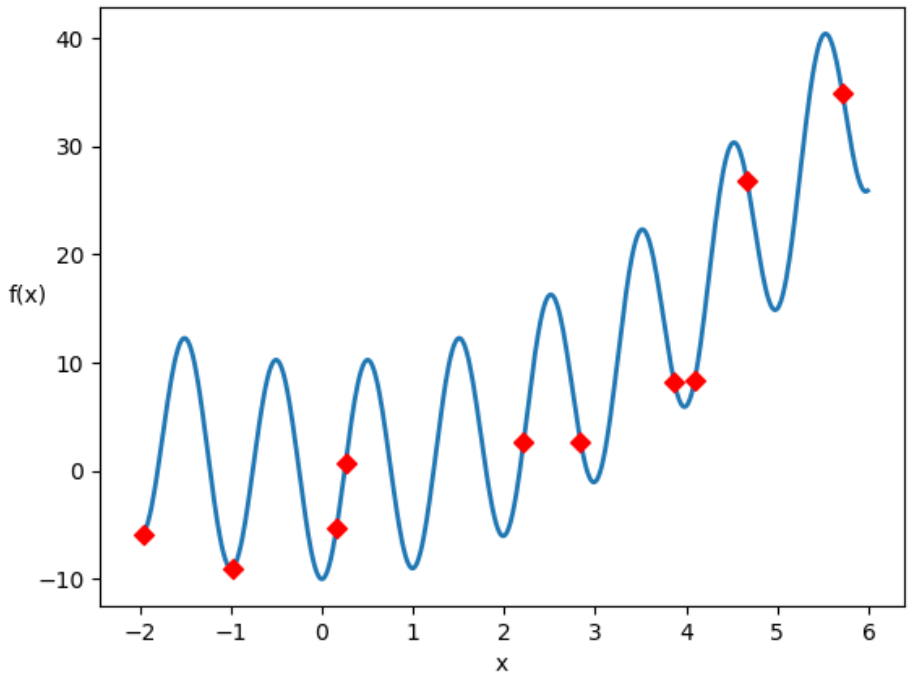

Figure 3.6: A numerical example for a simple Evolutionary Algorithm in one dimension. 10 solutions have been randomly generated and are shown here. In this example, the fitness is given by the $f(x)$ value.

the solution space, akin to genetic gene changes; the exact form of the optimisation depends on the algorithm. One full cycle where every individual gets evaluated and the variation operations have taken place is known as a generation. Under the abstract level, the mathematical details share very little with biological evolution. Instead, EA are more like black-box optimisation; a series of numbers form the solution, and these can be altered to produce the best single fitness value. The algorithms know nothing about the fitness function, only that it can be evaluated.

When evolutionary algorithms are used to optimise an ANN, each individual in the population is a neural network that is used as a solution. This approach is known as neuro-evolution. [127]. In the case of RL, the fitness evaluation for each individual is how well the network performs as a policy in the selected environment. Therefore, the

fitness function is directly related to the reward signal of the environment. Typically the game is played using the ANN, either until a terminal condition or the maximum number of timesteps has been reached, and the return is calculated for that trajectory. The fitness evaluation is then given by averaging several return values.

The type of variation operations available to the algorithm provides an internal distinction within the class of neuro-evolution algorithms. Some algorithms will only evolve the strength of the weights in a predefined neural network, but others may also evolve the architecture of the network. The latter are known as Topology and Weight Evolving Artificial Neural Network algorithmss (TWEANNs). Typically when using DNNs the network architecture is decided ahead of optimisation, leaving only the weights to be transformed. By changing the topology, TWEANNs are not directly comparable to successful RL algorithms in the literature that keep fixed networks. Consequently, TWEANNs are less common than methods that only alter the weights of the ANN.

Evolution Strategies

One subset of these optimisation algorithms are known as Evolutionary Strategies (ES) [128]. The premise is to hold a single set of parameters that form a solution, known as the parameter vector, and explore the solution space surrounding it. The parameter vector needs to have enough information to form the entire solution. By exploring the neighbouring solutions, the ES method can move the held parameter vector to an improved solution if there is one available. In the case of using ANNs as the solution, the parameter is the collection of weights and biases that would make up the entire ANN.

As with other evolutionary algorithms, ES generate a population of solutions to evaluate. To do this, the parameter vector is copied n_{pop} times to fill the population,

each copy is slightly varied from the original. The variations are introduced by adding generated Gaussian noise, denoted ϵ, to each individual in the population. The noise initially is between [0,1] but a scaling factor, σ can be added. This value can be used to alter how far the algorithm explores in solution space. A high σ increases the possible range of solutions to explore, but also decreases the density of solutions.

To optimise the solution, the parameter vector is moved towards the new members of the population that performed better in the fitness evaluation. Traditionally, the member of the population with the highest fitness score is held as the parameter vector for the next generation. The population is then cleared and the process repeats, using the new parameter vector to generate solutions for the next generation. The optimisation process continues either for a fixed number of generations or until the fitness value of the parameter vector is greater than some threshold value that indicates the problem is solved. This is formalised in Algorithm 3.2.

Algorithm 3.2 Simple Evolutionary Algorithm

```
1: procedure ES(n_pop, σ)
2:     Initialise policy vector w
3:     for i = 0,1,2,... do
4:         Sample ε_0, ε_1, ..., ε_{n_pop} ~ N(0, 1)
5:         Evaluate the population F_n = F(θ_t + σε_n)
6:         Set θ_{t+1} = θ_m where m is such that F_m = max(F_n)
7:         if F(θ_t) > Threshold then
8:             break
```

To test this algorithm, we will use it to obtain a maximum solution to the following equation.

$$f(x, y) = 20 + x^2 + y^2 - 10cos(2\pi x) + cos(2\pi y) \tag{3.5}$$

In this case, a solution is a two dimensional vector [x,y]. The solution space will cover [0,10] in both the x and y dimensions. The population will consist of 100 individuals that are generated with a σ of 2.5. Figure 3.7 shows the progress the algorithm makes

over 15 generations. The large white circles show the history of the parameter vector, the black circles show where each individual is located in parameter space, and the blue circle shows the solution in the current generation that has the highest fitness function.

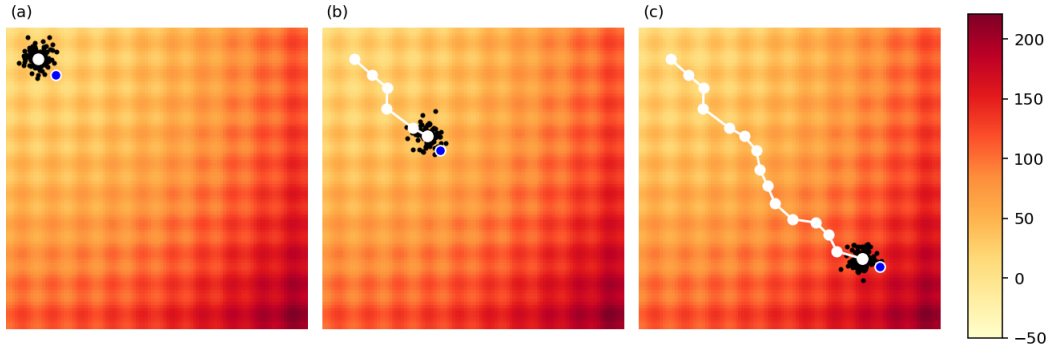

Figure 3.7: Numerical example for a simple Evolutionary Strategy. (a) Shows the initialisation step. (b) Each generation the parameter vector moves to the best solution found. (c) After 15 generations, the population is near the optimal solution.

As we can see, although the solution space has many local critical points, the algorithm is still moving towards the global maximum. This is due to the exploration of the population. The exploration rate in this scenario is large enough such that the population can explore beyond the local minima. The parameter vector is updated using just a single single solution from the population and the rest are discarded. This results in the path of optimisation often not being smooth and direct towards the global maxima. Given that only a finite number of discrete solutions are generated with random noise, there is stochasticity in the update step which results in the steps not always being optimal.

To reduce this affect, the researchers at OpenAI have refined the ES methodology such that it makes use of all the information in a generation [129]. The major difference to optimisation is how the parameter vector is updated; the key idea is to use

information from the entire population instead of just the member with the highest fitness. The new parameter vector is the weighted sum of every vector in the population, which is equivalent to taking estimating the gradient of the expected reward in parameter space. There is one additional hyperparameter that has been added, the learning rate. Denoted α, it is used to scale the weight change to the parameter vector, as seen in Algorithm 3.3 [129].

Algorithm 3.3 OpenAI Evolutionary Algorithm

procedure OpenES(a, b) ▷
    System Initialisation
    for i = 0,1,2,... **do**
        Sample $\epsilon_0, \epsilon_1, ..., \epsilon_{n_{pop}} \sim \mathcal{N}(0, 1)$
        Evaluate the population $F_n = F(\theta + \sigma\epsilon_n)$
        Set $\theta_{t+1} = \theta_t + \frac{\alpha}{n\sigma} \sum_{i=1}^{n} F_n \epsilon_n$
        if $F(\theta_t) >$ Threshold **then**
            break

Figure 3.8 shows how this algorithm performs on the two dimensional example, given by Equation 3.5. As before, the parameter vector is given by the white circle, and the black circles represent the population. An arrow was added to show the estimated gradient at the current point in the solution space. The optimisation path is smoother than the classic ES path as the new parameter vector takes information from all solutions. However, this method is not necessarily faster than the ES because the mutation rate controls the step size.

From this point, we will be using the OpenAI implementation when referring to ES. We will now use this method to optimise a neural network used for RL, in particular playing in the classic control environments, as discussed in section 3.1. For all the environments, the same network architecture was used. The network consisted of two hidden layers, each 128 nodes wide with the ReLU activation function applied on both. The population consisted of 100 individuals, the learning rate was set to

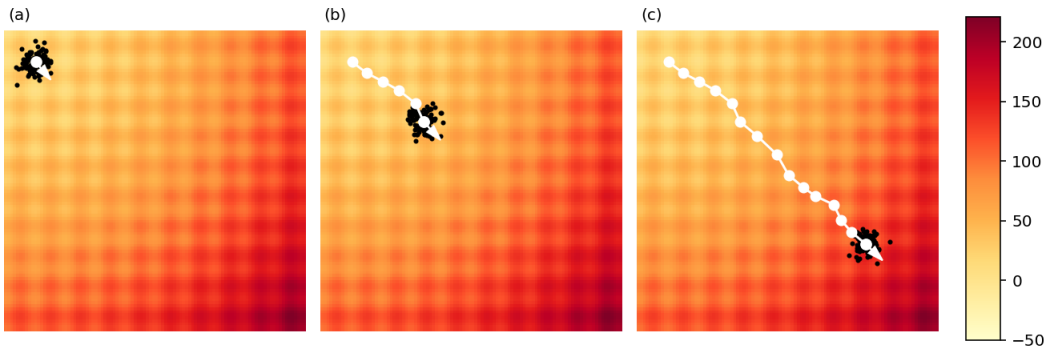

Figure 3.8: An example of using the OpenAI ES methodology to optimise a two-dimensional function. (a) The initial distribution of the population. (b) The parameter vector moves to a weighted average of the solutions in the last generation(c) Show the path taken over 15 generations.

0.05, and finally σ was set to 0.1. Figure 3.9 demonstrates the learning curves for the population on the Mountain-Car and the Acrobot environments. Both of these environments use the same reward function; -1 for each timestep played. The game ends when the terminal condition has been reached, or 200 timesteps have passed. The optimisation was ran 10 times using a different seed each time, and each policy would play 10 iterations of the environment. The training profile from the Cart-Pole environment is not shown in Figure 3.9 because the ES algorithm can reach the maximum possible score within 5 or less generations.

The score from the best policies found plateaued within 100 generations for the Mountain-Car environment, and 10 generations for the Acrobot environment. For both cases, the best performing individuals were on average reaching the goal within the first few generations. This result itself is impressive when comparing to the A3C method as seen in Figure 3.5. We see that for the Mountain-Car environment took **??** episodes to obtain any score that was above the minimum, whereas Figure **??** shows the ES algorithm achieving successful scores in the first few generations.

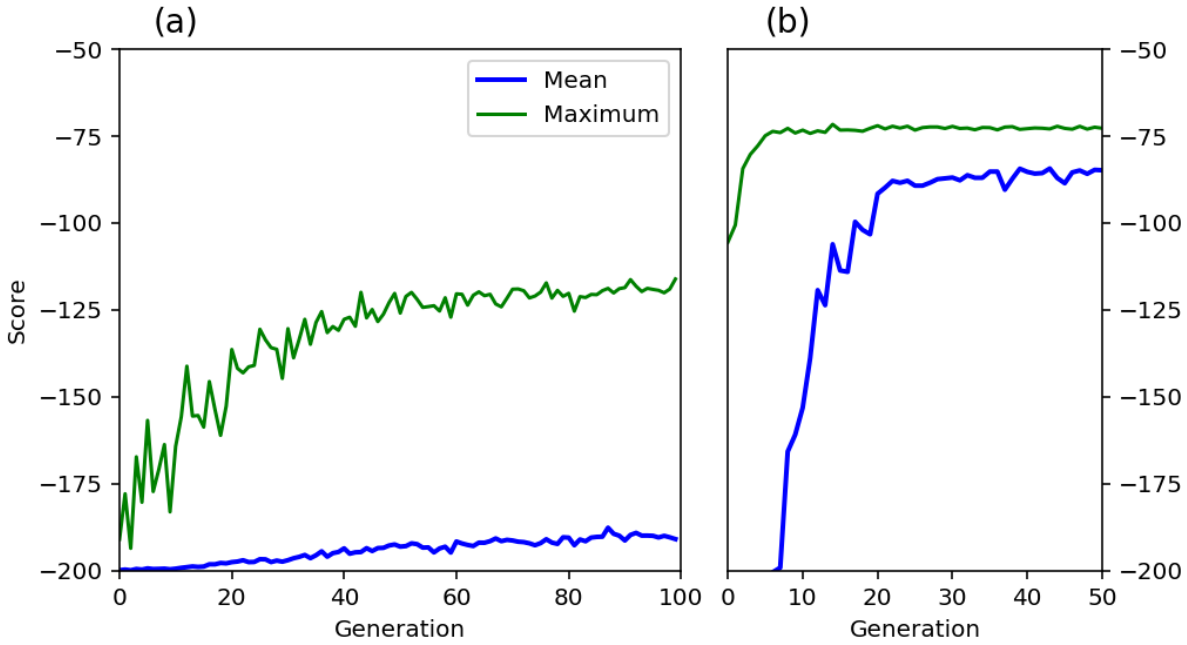

Figure 3.9: (a) Evolutionary strategies optimising an DNN to act in the MountainCar environment over 100 generations. (b) 50 generations of optimisation for the Acrobot environment.

Genetic Algorithms

Another class of algorithms that are seeing increased popularity in the RL literature are Genetic Algorithms (GAs) [130–132], which were inspired further by biological evolution and in particular Darwinian natural selection. In a genetic algorithm, we maintain a full population of different parameter vectors, unlike ES where only one is kept. To initialise the first generation, n_{pop} random solutions are drawn from across the solution space. These individuals are all evaluated using the fitness function and ranked based on performance. Those ranked highly are more likely to be used to create the next generation, hereby known as the parents of the next generation. The process known as selection takes the parents that are curated and places them into a 'mating pool'. This can be done either using a weighted probability based on the relative fitness of all solutions or by simply selected those which were ranked

highest. Enough need to be selected to fill the mating pool, the size of which is denoted as $n_{parents}$. The members of the population which are not selected to be in the mating pool are discarded. To refill the population for the next generation, we need to generate $n_{offspring}$ new solutions, where $n_{offspring} = n_{pop} - n_{parents}$. These new solutions, known as the offspring, are generated using variation operations.

Algorithm 3.4 Simple Genetic Algorithm

```
1: procedure GA(a, b)                                              ▷
2:     System Initialization
3:     for i = 0,1,2,... do
4:         Evaluate the population F_i = F(θ_n)
5:         if F(θ_n) > Threshold then
6:             break
7:         Remove the n_offspring lowest performing solutions from the population.
8:         for m = 0,1,2,...,n_offspring do
9:             Select from parents for reproduction.
10:            Generate new solution θ_m using variation operations.
11:        Add new solutions to the population
```

The variations can be split into two categories. If the operation has information exchanged between two or more members of the mating pool, then this is called crossover. If only one individual is used and gains new information, then this is known as mutation. Crossover first will select the individuals to combine from the mating pool. This can be done through shuffling the mating pool, effectively taking two or more random indices. For crossover of two individuals, the simplest form of crossover is known as single-point crossover. A random integer is generated in the range between 0 and the length of the parameter vector of the solution. A new solution is generated with all the parameters up to the index given by the generated integer taken from the first parent, and all the parameters after taken from the second. To slightly reduce computations, a second offspring can be created with the parent dependency flipped. To increase diversity, not all of the crossover operations will complete successfully.

After the parents are selected, there is a probability, p_{cross}, that crossover will take place. If not, then the parents are simply passed on to the mutation step as is. The mutation operator is also stochastic; when a solution is selected to be mutated, every parameter in the solution has a probability, p_{mutate} that it will be changed. To change the parameter, a number, ϵ is drawn from a random normal distribution, scaled using some predefined factor, σ, and added to the parameter. If every parameter in the solution is mutated according to the probability p_{mutate}, then the offspring shares no parameters with its parent.

In the work done in this thesis, crossover was not used. This method of using a GA with no crossover is the same as that used by Such et al (2018) [122]. Instead, the population was refilled by purely by applying the mutation operation to individuals in the mating pool. $n_{pop} - n_{parents}$ are randomly selected with replacement from the mating pool, mutated and added to the offspring. Without crossover, it is very similar to the ES, the major difference being the population that is maintained between generations. From the work done using the ES methodology, we know that using a mutation only method is viable. Also, let us consider the effect of crossover in a DNN. The output of any node past the input in a feed-forward network depends heavily on the weights of the connections into it. If we were to swap out an entire layer of weights, then the next layer of nodes will likely need retraining to calibrate for the new inputs. Likewise, with any collection of nodes, the network has a dependency on them as a whole. To illustrate the GA method, we have used it to find the global maximum in Equation 3.5. Figure 3.10 shows the population after initialisation, 1 generation and 5 generations in (a), (b), and (c) respectfully. As with the previous example, 100 solutions are generated in a two-dimensional solution space. After each solution is evaluated, the 25 fittest are kept as parents for the next generation. Therefore, 75 offspring are generated using the mutation operation with p_{mutate} set to 0.75. The

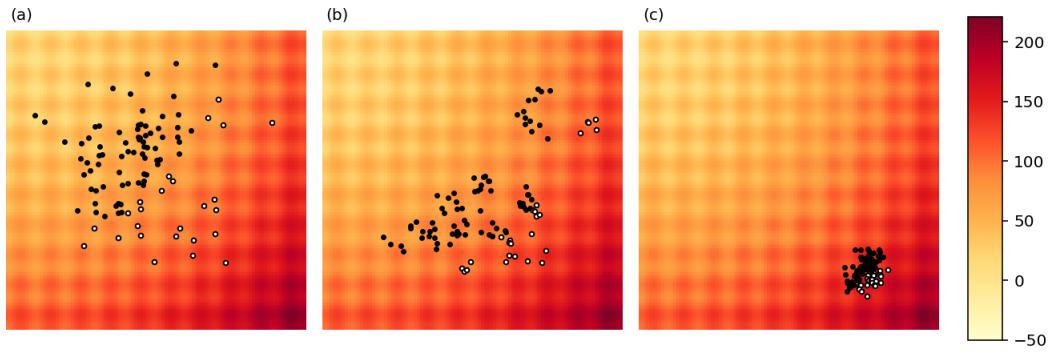

Figure 3.10: Numerical example for a simple Genetic Algorithm. The white circles are the best performing individuals that are kept for the next generation. (a) The initial spread of solutions. (b) In the next generation, there are no solutions in the top-left quadrant where the fitness is lowest. (c) The population has now collapsed around a smaller area and will move in a similar manner to the ES method.

white circles are those that will be kept for the next generation, while the black circles will be replaced in the population.

Due to the initial spread of solutions, the GA is much quicker to locate the area of high fitness; it takes 5 generations compared to 15. The ES algorithm depends very heavily on the initial parameter vector. As we saw in Figure 3.8, if the parameter vector is located in an area of low fitness, then the algorithm will need many steps to find the optimal solution.

We also tested the GA algorithm on the classic control environments. As with the ES the networks to be optimised consisted of two hidden layers, 128 nodes wide using the ReLU activation. The population consisted of 100 individuals, the mutation percentage is set to 0.75, and finally, σ was set to 0.1. Figure 3.11 shows the results for the Mountain-Car and the Acrobot environments. The results were obtained using 10 random seeds, and each policy would play 10 iterations of the environment.

As with the ES method, the score from the best policies plateaued within 10

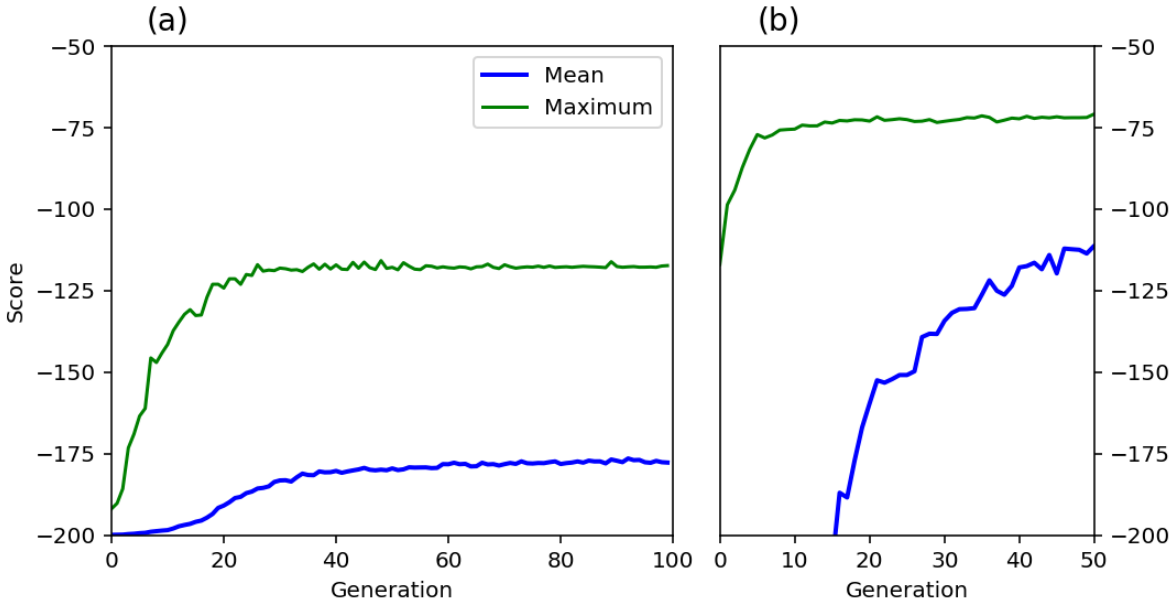

Figure 3.11: (a) Score in the Mountain-Car environment using a DNN policy that was optimised using a genetic algorithm over 100 generations. (b) The score over 50 generations for a likewise policy in the Acrobot environment.

generations for the Acrobot environment. However, we do see a slight decrease in training time on the Mountain-Car environment if using a GA. Figure 3.9 shows the score does not reach the highest score until approximately 50 generations in. Comparing this to Figure 3.11 where the score remains constant from around the 25th generation onward. It is likely that this improvement stems from the ability of the GA to explore a larger range of the state space from initialisation. The increases seen with the score curve also appears to be smoother when using this method. One factor we can attribute this to is the GA keeps the best performing members of its population for the next generation, whereas the ES generates an entirely new population.

NeuroEvolution of Augmenting Topologies

The last EA that we shall look at is an extension on a GA, known as NeuroEvolution of Augmenting Topologies (NEAT) [133]. Unlike the previous two algorithms, NEAT was developed specifically for evolving ANNs. It follows the same method as a GA seen in Algorithm 3.4; creating a population of individuals where the strong are more likely to breed and performs variation operations to create new solutions. Where NEAT differs is the mutations that are available. As aforementioned, NEAT is designed for ANNs and so the mutations are designed to create varied neural networks as solutions. Both methods discussed previously assume a fixed neural network architecture and only altered the weights. NEAT on the other hand has mutation operations available which will alter the structure available, as well as the weights.

Before discussing the variation operations available, it is important to know how the population is initialised. As the structure of the network is being altered, the designer does not need to decide the network architecture beforehand. Instead, the network starts with the most minimal topology possible; a fully connected network that joins the input layer to the output layer with no hidden nodes. This ensures that the algorithm searches the lowest dimensional weight space first before moving onto more complex solutions. Figure 3.12a shows the initial network for an environment that has a state space of size 3 and 2 actions. By starting with the lowest dimension solution, NEAT aims to keep the solutions minimal over all generations. There are four types of mutation operation available, as seen in Figure 3.12. The first operation is weight mutation. This is done in the same way or similar to the weight mutations described for a GA. Each weight in the ANN is either altered or left alone according to some probability, denoted here as p_{weight}. In our implementation, if a mutation is set to happen, a random number is drawn from a Gaussian distribution, scaled using

scaling factor σ, and added to the weight value.

The second form of mutation is to add a node. To ensure the node is in a valid place, a current connection in the network is randomly selected. We'll denote the node that starts this connection as n_x, the node at the end of the connection n_y, and the new node n_z. To add n_z into the network, the connection from n_x to n_y is copied but links n_x to n_z. The original connection is between n_x and n_y is disabled. A new connection is added between n_z and n_y with weight 1. By doing so, the signal that reaches n_y does not change. This is to encourage diversity in the solutions; if the weight is changed, then the new solution may be discarded in the next generation, even if adding a node is beneficial for the problem. There is also a mutation that just adds a connection. A connection cannot be added between two input nodes or two output nodes, nor can one be added between two nodes that already share a connection. The weight for the new connection is drawn from the same distribution as the weights for the initial connections. Finally, the mutation operation can also disable a connection. To do so, a random connection is selected from all connections in the network. When a connection is disabled, the weight value remains the same, but the weight and corresponding input are not used in the node calculation. If the selected connection is already disabled, then it is re-enabled.

These networks are feed-forward, which means the connections between nodes pass signals only one way. To ensure this is the case, we add a layer variable to each node. Input nodes are on layer 0; any node after is assigned to the layer one higher than the highest layer of the input nodes. When a connection is added, if the two layer variables are different, the connection feeds from the lower layer to the higher layer. If a connection is added between two hidden nodes on the same layer, a direction is randomly assigned and the layer variables are recalculated.

Once again, we tested the NEAT algorithm on the classic control environments. As

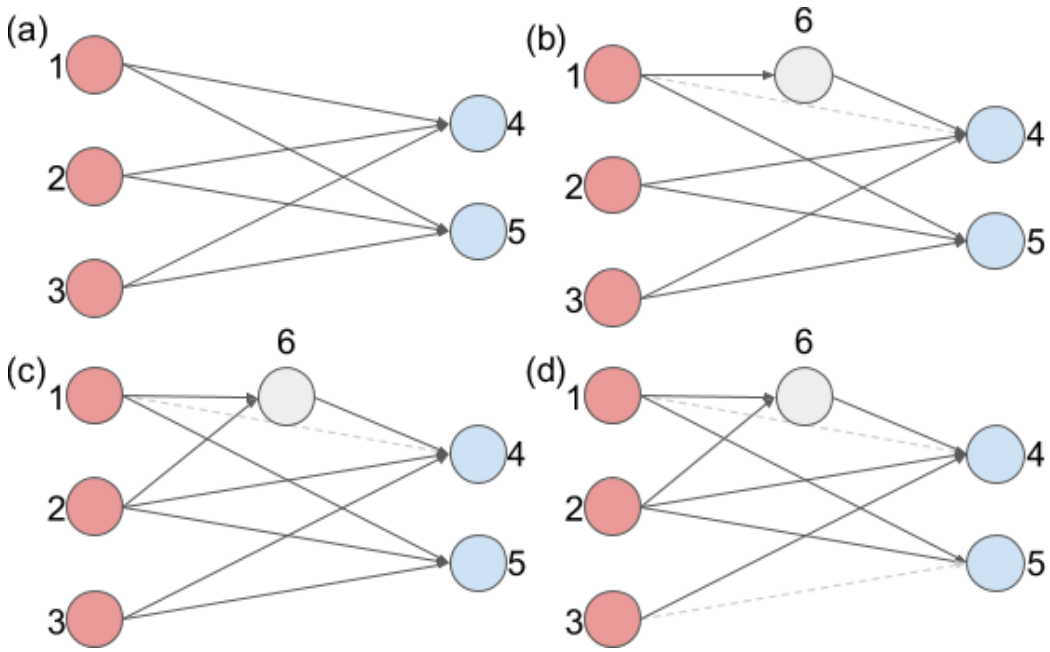

Figure 3.12: (a) The initial configuration when using the NEAT algorithm (b) Shows the node addition mutation. A node was added along the connection between nodes 1 and 4. The original connection remains but is disabled. (c) Shows the mutation that adds a connection. A connection was added to join nodes 2 and 6 (d) Shows the disable/enable connection mutation. The connection joining 3 and 5 was selected and disabled.

with the GA test, the population consisted of 100 individuals and 75 were replaced each generation. The probability of each mutation occuring is independent of the other mutations. The weight change mutation had a 75% probability of occurring, and the other three all had a 25% probability of occurring. When the weights were mutated, the mutation rate $\sigma = 0.1$. As with the ES and GA algorithms, the networks could solve the Cart-Pole environment almost immediately. However, when using the NEAT algorithm, the ANNs at initialisation have no hidden nodes; the signal is passed from input to output through a single layer of weights. This means that the Cart-Pole environment can be solved by a neural network policy that contains no hidden nodes.

Figure 3.13a shows the training curve for the Mountain-Car environment. The

highest score observed in the population plateaus at a similar value to the GA test (Figure 3.11a). However, the average performance from all individuals in the population is considerable higher when using the NEAT algorithm. This can be attributed to how the ANNs are initialised. The neural networks in the population are all created to be as simple as possible from the start, and have only enough weights to connect the input nodes to the output nodes. When the weights are mutated, an ANN with few connections will vary less than an ANN with considerable more connections. This results in children that are more similar to the parents in the NEAT population compared to using a GA with a more complex ANN. With this in mind, we can study the structure of the best performing individuals that form from the NEAT mutations. Figure 3.13b shows an example of one of the best performing ANNs on the Mountain-Car environment. This particular individual was taken from generation 40, approximately when the maximum score begins to plateau. Starting off with a minimal network, the ANN does not have many generations to alter the topology successfully, especially as all mutations other than weight changes have only a 25% chance of occurring. We see only two hidden nodes have been added to the initial topology. With this configuration, the ANN was able to perform comparably to the DNN architecture optimised with a GA, which had 128 nodes in two layers.

The last environment we tested the NEAT algorithm on is the Acrobot environment. Similarly to the Cart-Pole environment, we expected this algorithm to be solved very quickly, just like we had seen when using the ES and GA methods. From Figure 3.14a, we can see the maximum score of the population is nearly at its highest from initialisation. The maximum score when using the NEAT algorithm is about the same as the previous EAs that we tested. As we saw with the Mountain-Car environment, the average score for the generation is much higher than when using a GA. This is likely for the same reason as with Mountain-Car, that less variation

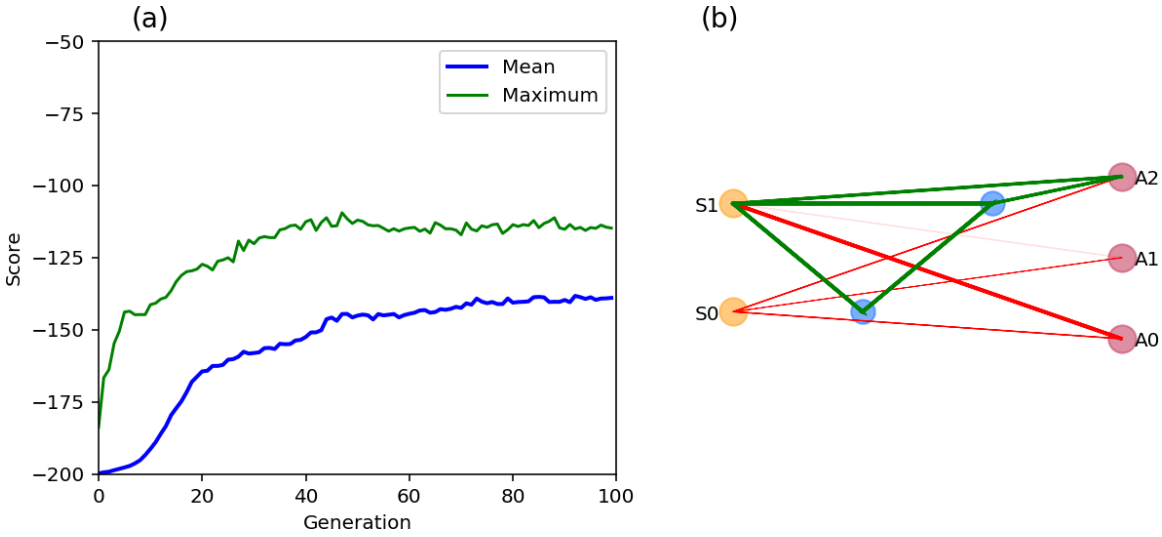

Figure 3.13: (a) The Score during optimisation of a policy acting in the Mountain-Car environment. (b) An example of an ANN produced by the NEAT algorithm. This network was the top performing after the 40^{th} generation.

can occur during mutation because there are less weights. We can also visualise the simplicity of the solutions found using the NEAT algorithm. Figure 3.14b shows one of the best scoring ANNs from the tenth generation. By this generation, the score had plateaued and the neural network policy succeeds in the environment without any hidden nodes. Once again, the NEAT algorithm shows that the solution is much simpler than initially estimated.

NEAT has seen a steady increase of improvements and reimaginations since its inception. The original author of NEAT implemented an extension that allows the evolution of solutions in real time rather than in generations [134]. It does so by giving each individual a timer. When time is up, the solution is evaluated and removed if it performed poorly. At this point, it is replaced by a child from high-fitness parents. Most recently, a variation of NEAT without the weight mutations has been developed. Referred to as Weight Agnostic Neural Networks [135], these networks share a fixed weight value between all connections. To optimise the network, the authors used operations to change the topology of the network and the activation function at each

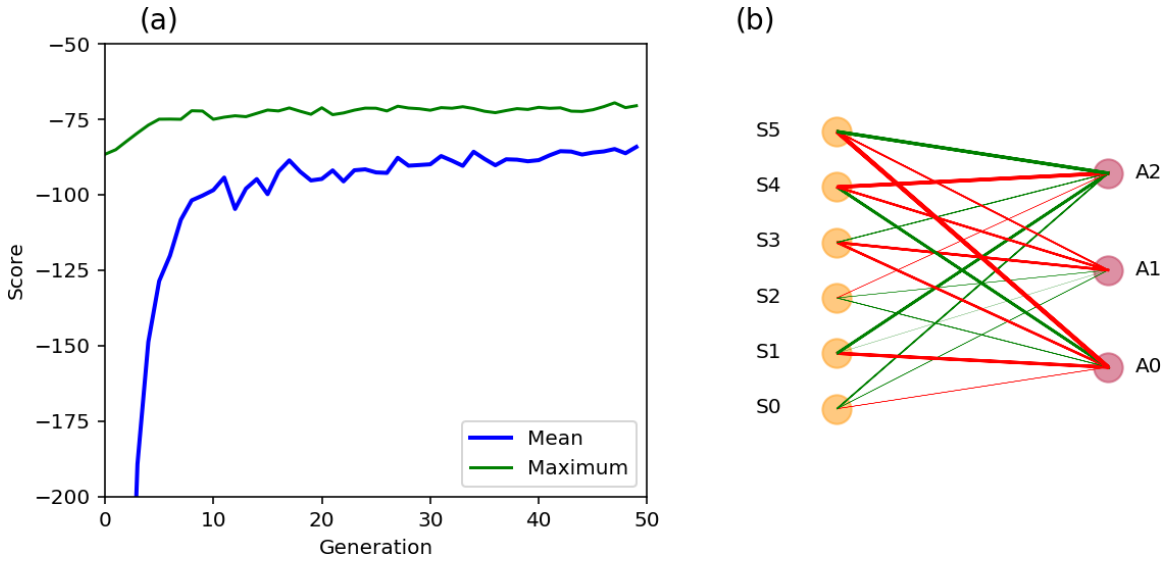

Figure 3.14: (a) The Score during optimisation of a policy acting in the Acrobot environment. (b) An example of a network produced that could complete the goal. This particular network is from the tenth generation.

node.

3.3 Curriculum Learning

In this section, we will introduce the concept of transferring knowledge between ML models, and how this technique can be used to improve the training time on difficult problems by first learning how to solve an easier problem. Having a series of concepts to learn, often ordered by complexity, is a well-explored idea in the context of human learning. In schools, teachers will create a curriculum for their students that will gradually introduce more complex topics, allowing the students to exploit knowledge that has been learnt before. This notion of learning via a curriculum is now being explored more prevalently in the field of ML to learn complex assignments. We will study some examples of curriculum learning in RL that demonstrate how an agent can use prior knowledge of a task on a similar, but more difficult problem. Finally, we will explore the concept of curriculum learning in the literature and how the field has progressed in recent years.

Many ML methods make the assumption that the task the model will be tested on is the same task it was trained on. For example, testing of an SL model would use a test dataset taken from the same distribution as the training dataset, or the testing of an RL agent would occur in the same environment that it was trained in. Should the task be changed, many models would need to be retrained from initialisation with data from the new task. For simulations, the cost to retrain the model may be minimal, just the time needed to recollect samples. However, in real-world applications, it may be expensive or impractical to recollect the necessary training data for building the models again. Transferring the learnt knowledge between tasks is needed to reduce the burden of retraining models [14, 136].

Transfer learning refers to the learning that is one in one situation being exploited to improve generalisation in another. There are no constraints on the distribution of

data, the domain of the problem, or the task to be completed. However, there is an assumption that many of the factors that explain variations in the first situation are relevant in new situations.

Let's look at an example regarding image classification. The initial task is to learn to distinguish between images from a set of categories based on one dataset. Once trained on this set, using transfer learning we can apply the learned knowledge to another dataset to quickly learn a different set of categories. This is possible with image classification because many objects share the same low-level notions such as edges, or lines forming shapes. Thus knowing these representations helps to quickly generalise for the new set. Transferring knowledge may be used in SL where the task has only a few training examples, but there are similar tasks where training examples are readily available. The larger training set would be used to learn the low-level features and then this knowledge would be applied to the target set. This example shows transfer learning taking similar inputs and learning a generalised representation that can be used for multiple distributions of the input.

Curriculum learning is a particular form of transfer learning, where the knowledge is transferred from one task to a similar but more difficult task. Continuing with the example of image classification above, consider the task of classifying images of shapes, such as rectangles, ellipses, and triangles. An easier task to train on first would be with shapes of less variability, such as squares, circles, and equilateral triangles. Once the model can solve this task, you could use this knowledge for faster training of the first. This one of the tasks used to showcase curriculum learning in Bengio et al. (2009) [137]. In this work, an ANN was trained for a fixed amount of epochs on the two problems as described. They showed that the classification error was much lower when the ANN had time to train on the simpler task first, before being applied to the harder problem.

Curriculum learning for RL problems was introduced in 2016 by Narvekar et al. [138]. Before then, transfer learning had been gaining popularity and was being used in RL paradigms [139–141]. There are many ways to create similar RL environments with varying difficulties, such as change the size of the action space, the state complexity, or altering the reward function to induce or remove sparsity. Complex environments often require multiple tasks to be achieve and so can often be broken up into subtasks which can all be learnt independently.

Let us consider the Mountain-Car problem as discussed in Section 3.1. As aforementioned, the car can not simply accelerate in the direction towards the target and successfully make it up the hill. The need to learn a strategy without feedback is what makes this problem difficult. However, with a shallow hill, the car can easily make it to the top just by moving in that direction. Therefore, we can pose the Mountain-Car environment as a transfer learning problem. We hypothesised that if the agent learns how to achieve the objective on a shallow hill, then it could learn how to do so on the original problem quicker compared to without this strategy.

To test this, we created a version of the Mountain-Car OpenAI gym environment that accepts a height parameter on initialisation. This height parameter, h, defines the steepness of the hill for that iteration of the environment. h is bound in the range [0,1] where 0 is a flat path, and 1 is the original hill height. The height at any point in the valley is given by Equation 3.6. This function is used in the original environment to define the hill height; the only difference is that we introduced h as a scaling factor. Figure 3.15 illustrates the effect of altering the scaling factor. Once the hill height reaches 0.6, the car needs to gain momentum before attempting to move up the hill.

$$f(x) = h \times 0.45sin(3x) + 0.55 \tag{3.6}$$

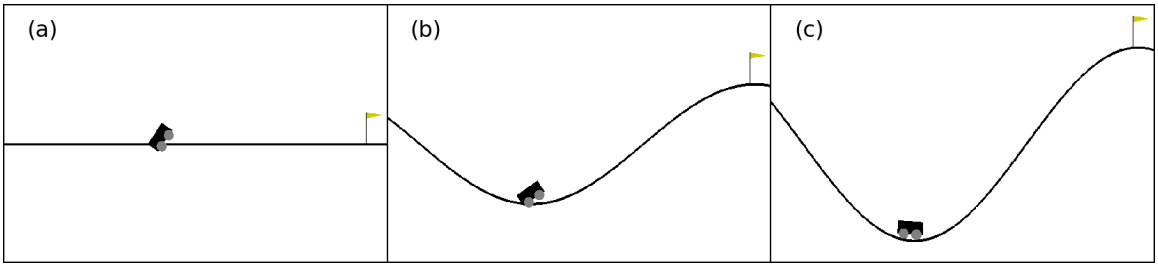

Figure 3.15: The Mountain-Car environment with a variable hill height. (a) The first initialisation of the environment with a hill height of 0. (b) The hill height is set to 0.6; it is at this point where just moving right will not work at all for any initialisation. (c) Shows the final stage where the hill height is 1. This environment is identical to the original.

For all of the following experiments, the agents used ANN policies that have two hidden layers, each 128 nodes wide, utilising the ReLU activation function. The networks were optimised using a GA that hosts a population size of 100, a mutation chance of 75%, and a σ value of 0.1. As the games have random elements in the initial state, each policy was tested 10 times per generations using different seeds, and we run the optimisation process with 5 initialisations.

To test the hypothesis, we trained the DNN first on the environment with a shallow hill, h = 0.5, for a fixed number of generations, t_c. At this height, the car could achieve the goal just by accelerating towards the target. After t_c generations, the environment switched back to the original and the policy continued to train. From Section 3.2 we know that using a GA that the best policy begins to plateau after approximately 25 generations. Therefore the values of t_c needed to be lower than this for us to be able to see an decrease in training time. Figure 3.16 shows the score curves using t_c values of [1,5,10]. Note that while being trained on the shallow hill the score was higher because it takes less steps to reach the goal. We then expected to see a dip in performance as the agent adjusts to more difficult environment. The results are also compared to those as obtained in Section 3.2 when no curriculum was used.

We can see that even just by adding a single generation of training on the simpler

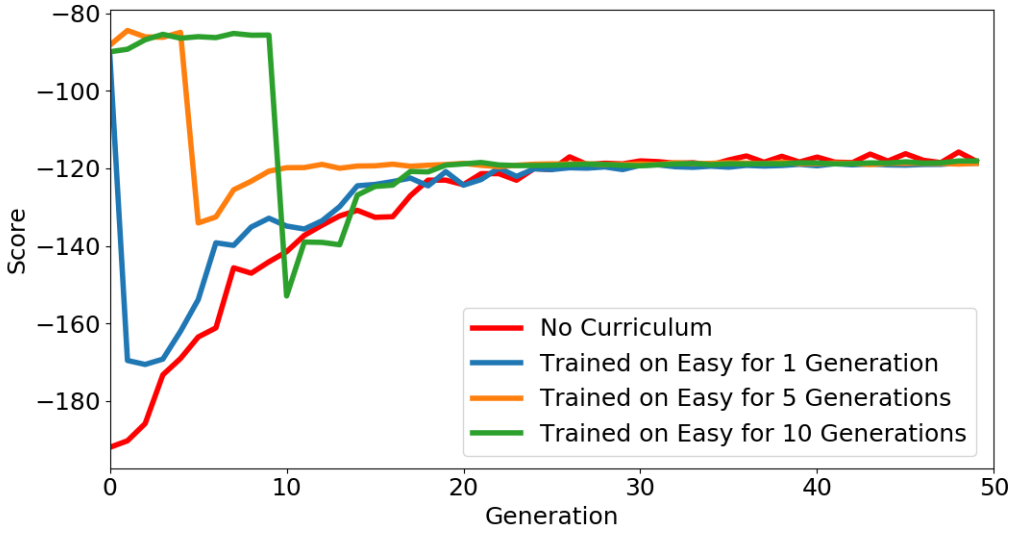

Figure 3.16: The score from each generation on the Mountain-Car environment with a variable valley depth. The curriculum trains the agent on an shallow hill, and switches back to the original after a fixed number of generations. Three variants of the curriculum variants are shown.

environment the performance is improved compared to when no curriculum was used. The results are best when the switch between environments occurs at 5 generations. By doing so the score begins to plateau around 10 generations, which is the best performance of all the tests. When the switch occurs 10 generations in, the agent that does not use a curriculum has already passed it in performance.

We can also apply this technique to the Acrobot environment. The goal for this environment is to maneuver the lower pendulum link to a target height. Therefore, to lower the difficulty, we can lower the target height. This is slightly different to the change made to the Mountain-Car environment because we are not changing the fundamental environment at all, only the success condition. In the original version, the target height is the length of one link above the highest joint. We will be referring to this height as a value of h = 1. Shown in Figure 3.17 are the environments with h = -1 and 0 compared to the original. The lowest value of h we used is -1, which in

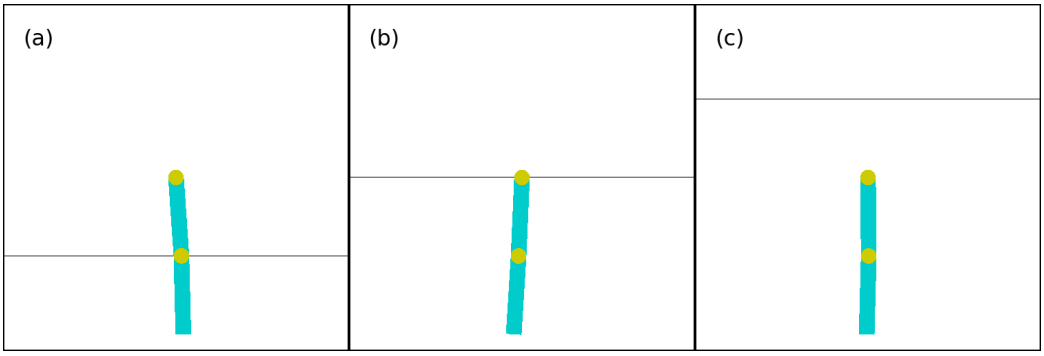

Figure 3.17: The Acrobot environment initialised with different target height values. (a) Initialised with a target of link length below the centre. (b) The height of the target is set to be at the centre. (c) Shows the final stage where the target is one link length above the centre point. This environment is identical to the original.

the environment is one link length below the highest joint; h = 0 corresponds to the target height being the same height as the reference joint.

As we saw in Figure 3.11b, a GA could optimise a policy for the Acrobot environment very quickly, within 10 generations. For an environment such as this, curriculum learning may not be necessary as there is not much room for improvement. However, we can show that this technique still does decrease training time. Figure 3.18 shows the results of using the step increase of difficulty, with $t_c = 1,2,3$. As with the Mountain-Car environment, we expected a drop in score when the difficulty is increase. When one generation of solutions is optimised on the easier environment before being switched, we observe the performance is actually worse than if no curriculum is used. However, for the cases where $t_c = 2$ or 3, when the difficulty is increased the optimal policy immediately plateaus. Even for an environment which converges after 10 generations, adding a simple curriculum can be shown to shorten the training time.

Using a step function in difficulty is a limited approach to adding a curriculum. For the classic control environments, we already know roughly how many generations is appropriate for training. Therefore we can adjust at what generation the increase

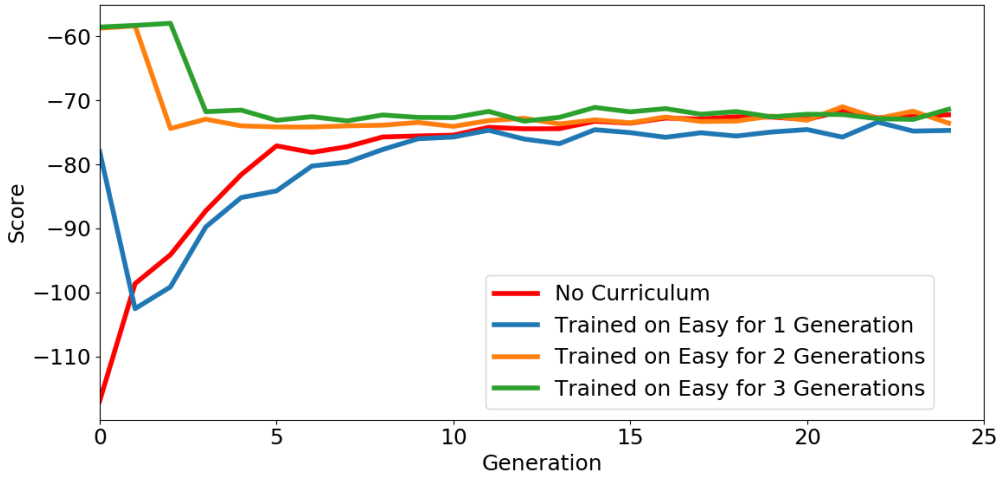

Figure 3.18: The score from the best policy of each generation trained on the Acrobot environment. The difficulty was increased after 1,2, or 3 generations and compared to the results if no curriculum was used.

in difficulty occurs at. To use this method on environments where this knowledge is not available, either the researcher must estimate an appropriate value or use another method. It is important to add that the transferring of knowledge can happen more than once. Typically the curriculum is sequence of tasks that gradually increase in difficulty or introduce new elements [138]. Each stage of the curriculum should be appropriate for the current ability of the model. If too easy or too hard, the model will fail to learn any new strategies.

Finding the best curriculum, i.e. the optimal sequence of tasks, is the one of the biggest areas of research regarding curriculum learning. The tasks must provide a suitable jump in difficulty and ideally creating the task would not be particularly work intensive. Many transfer and curriculum learning methods assume the series of tasks are provided, which requires human knowledge in advance to create the tasks. There is ongoing research in automatic task creation which aims to produce a series of agent-specific sub-tasks automatically [138, 142]. For example, one set of methods to create sub-tasks is to identify when the agent makes a mistake and then focus on

learning a solution to that mistake. This process is known as Mistake Learning. The optimal policy is not known in advance, and so mistakes are not obvious. However, even without knowing the best action to take, for many environments we can assume that any action which directly results in an unsuccessful termination state is a mistake. From this point, the algorithm reverses the state by a fixed number of time-steps and trains on this area of the state-space.

To effectively use curriculum learning, the training tasks must also be sequenced such that knowledge is kept from past experience but also updated when new knowledge is learnt. In the case of mistake learning, the mistake sub-tasks are learnt when the mistake is discovered. However, if the environment is comprised of sub-tasks that can be learnt independently, or tasks of varying difficulty, then assigning a training schedule for the agent is non-trivial. Using ANNs to choose what tasks the agent trains on is a relatively new concept being explored in the literature [143–145]. The terminology varies with the author, but the common theme is to have a primary DNN learn to complete the goal of the environment. To do so, it must train one or more subordinate DNNs to achieve specific sub-tasks. Using an ANN as such has been shown to be an effective method of selecting a curriculum without manually assigning tasks to be learnt.

Scientific Method Application

The success of deep-RL has led to scientists using pre-trained RL agents to control physical robots. Polices that are obtained purely in simulated environments can be trained to account for the additional physics of the real-world [146]. In the past, model-based RL has been used when training is done in a real-world environment because model-free RL is typically far too sample inefficient. However, model-based RL is highly unstable for problems with complex dynamics and optimisation of the policy can be influenced heavily by the model [147–149]. Given the success of model-free RL, we would like to continue to develop methods for these algorithms to be utilised in the real-world.

The real-world application we will be focusing on is chemical laboratories. AI-assisted laboratories have the potential to dramatically increase scientific discovery by intelligently automating procedures and experiments in a series of environments [150]. When conducting experiments in these environments, measurements can often be expensive, either due to the resources used or the time that is taken. Due to this, measurements are typically taken intermittently or only when exploring new conditions. If we were to use RL to optimise a laboratory experiment, it would ideally act in a similar manner. Therefore, the RL agent would need to act without access to perfect information and balance the cost of observation with the need for new data.

The environments which we have looked at so far can all be formalised as MDPs. However, in for an agent to act without perfect information, the state does not contain all the information required to make decisions and the MDP formalisation does not apply. Instead, a problem that requires an agent to act with an uncertain state is more similar to a Partially Observable Markov Decision Process (POMDP). The difference

between an MDP and a POMDP is that an agent cannot observe the underlying state in a POMDP. Instead, the agent acts on a belief state, which is typically a probability distribution over all states available to the agent. While a POMDP shares many characteristics with our problem statement, the access to observations is what sets them apart. When the agent in a POMDP takes an action, it receives an observation from the environment based on the conditional observation probabilities defined in the environment. We would like the observations to be decided by the agent instead of the environment. Doing so reflects the scientific application of the RL algorithm that is the end goal for this project.

In the following section, we show how an agent can build an internal hypothesis of its environment, using experience from past measurements, that it can then act on. To achieve success, the agent must extrapolate from the known hypothesis, conduct a test in order to obtain new data, and then base future decisions on those results. This is a well-known process: the scientific method. We then show how our newly developed method can be used to learn the dynamics of physics-based models and exploit the knowledge gained to achieve a given objective with a measurable confidence.

3.4 Creating a Model

To begin, we will discuss tools that we can use to model an environment. There are some important characteristics that a model must have for it to be useful. Firstly, the model should be accurate near observations that we know to be truthful. This is a necessity when using this model to plan appropriate actions. Furthermore, the model needs to be able to be updated when new measurements are taken, increasing the information in the model. Secondly, the model needs to interpolate between the known measurements as we would expect only intermittent observations to be

available in a laboratory environment. Finally, the model needs to incorporate a measure of confidence in its predictions. These measurements will be used to judge when the model needs more information.

3.4.1 Neural Networks

Neural networks are not probabilistic in nature, so when they make a prediction, there is no measure of uncertainty. Therefore every prediction, regardless of the accuracy, is taken with equal confidence. This is problematic in a wide variety of applications across ML, particularly in cases where mistakes have serious consequences. Consider an autonomous car, for example, that uses a DNN to detect objects in front of the vehicle [85,86]. Poor weather conditions or a new environment may cause the network to misjudge what action the car should take, a mistake which could perhaps prove fatal for the passengers or other road users. Some other examples of environments with serious consequences are medical screening [151–153], or pipeline monitoring [154]. In these environments, predictions with high uncertainty should be reviewed in an effort to reduce risk. In the autonomous car example, in high-uncertainty situations, the car should have the human driver take over.

To explore how the uncertainty of an ANN prediction could be used, let us consider a classification problem. In this example, suppose an ANN has been trained to distinguish between images of two objects. If the input to the ANN is a picture of either of these two objects, then it should be able to label them correctly. We'd expect the prediction, in this case, to be made with high confidence, because the neural network saw many examples of these objects during training. However, if the input to the ANN is an image of a previously unseen object, then the prediction will be inaccurate. Therefore, this prediction should be made with low confidence, allowing for a follow-up action to be taken if needed. When an ANN is exposed to

data not from the training distribution, it must extrapolate to make a prediction, which will often give unpredictable results.

Ensemble Averaging

One method to estimate the uncertainty of an ANN is to use multiple copies of the ANN, all making the same prediction. This method is known as ensembling. The ANNs in the ensemble are initialised and trained individually on the same dataset. The key idea is that predictions from all of the neural networks will converge to a similar result around the training data. Further away from the observed data, the results will be more diverse because the networks have not been trained on this data. For a single query, a prediction is sampled from each model, and thereby an average prediction and the corresponding variance can be calculated. The mean prediction is taken as the predicted value of the model and the variance as a metric of model uncertainty. If N different models are in the ensemble, and each model outputs a prediction $y_{i\in N}(x)$, for a single input x the ensemble prediction and variance are given by:

$$\begin{aligned} \bar{y}(x) &= \frac{1}{N}\sum_{i=1}^{N} y_i(x) \\ \sigma_x^2 &= \frac{1}{N}\sum_{i=1}^{N} (\bar{y}(x) - y_i(x))^2 \end{aligned} \tag{3.7}$$

A simple ensemble is straightforward to implement; if one neural network can be used for the model, then each of the additional ANNs are trained identically. Often identical architecture is used for each of the ANNs. We have tested this methodology for a DNN applied to two regression problems. For the following tests, all the ANNs in the ensemble will consist of two hidden layers that are both 128 nodes wide, each

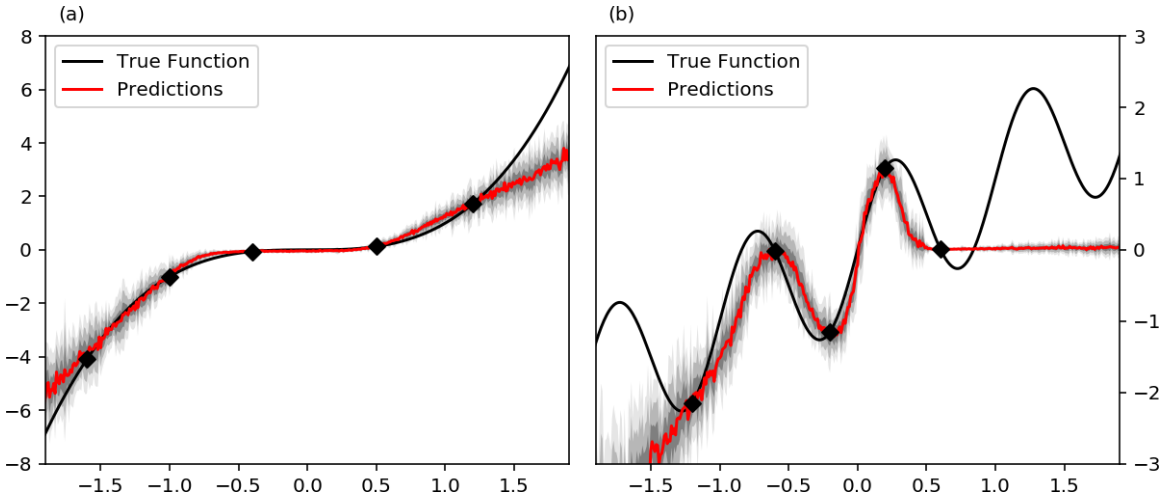

Figure 3.19: This figure shows a neural network ensemble predicting the outputs of learnt functions. On both figures, the observations are shown by black diamonds, and the uncertainty is indicated by the shaded area. (a) The ground truth function was $y = x^3$. (b) The ground truth functions was $y = sin(x) + x$.

with the ReLU activation function. The ensemble consisted of 10 DNNs with different initialisation but all trained on the same data. Figure 3.19 shows the ensemble method applied to two functions: $y = x^3$ and $y = sin(x) + x$.

We can see from Figure 3.19a that the network can estimate the function $y = x^3$ well within the range [-1,1]. In this range, the function is bounded by observed training data. Beyond the training data, the model does not extrapolate well. The prediction tends away from the true function, but the uncertainty is also increasing as the model moves away from the training data. Having low confidence away from the training distribution is a beneficial feature for our problem because it indicates a need for more data. Figure 3.19b shows the ensemble method has similar behaviour on $y = sin(x) + x$. The model is not completely accurate compared to the ground truth, but considering only five training points were used, it is a reasonable representation. In the range of the data points, the model captures the increasing periodic trend of the function. However, once again, the ensemble method performs poorly beyond the training data . The ensemble predicts that all outputs beyond $x = 0.5$ are

approximately zero with low uncertainty. This behaviour is particularly interesting as this trend is not seen anywhere else in the model.

Additions to the ensemble method have been presented in the literature. One such update to the model is to use a weighted average [155]. To do so, each ANN is assigned a scalar weight value. Assigning the best weight values for each network is then an optimisation problem, one which can be solved with an additional ANN. The main downside with all ensemble methods is that the training cost scales proportionally to the size of the ensemble. For complex models, where the number of parameters can be in the order of millions, the time taken to train multiple ANNs becomes unwieldy.

Monte-Carlo Dropout

An alternative method of measuring confidence from an ANN is known as Monte-Carlo Dropout (MC Dropout) [156]. This algorithm only requires a single neural network that has been trained with dropout enabled, meaning it is considerably quicker than training an ensemble. Dropout is a regularisation technique that randomly disables nodes in the neural network to prevent overfitting. In typically use-cases, dropout is only applied during training, but when using MC Dropout, we also use dropout during inference as well. By randomly eliminating nodes from the model, the prediction is no longer deterministic and relies on which nodes remain. Therefore, if we make several predictions, each with a different collection of nodes disabled in the network, we will obtain a collection of slightly different predictions. Therefore, by taking a large number of predictions, an average prediction and standard deviation can be calculated.

Figure 3.20 shows the results of using the MC Dropout method. The results are extremely similar to the results from the ensemble method, shown in Figure 3.19. We see almost identical predictions, including the extrapolation behaviour. This result is

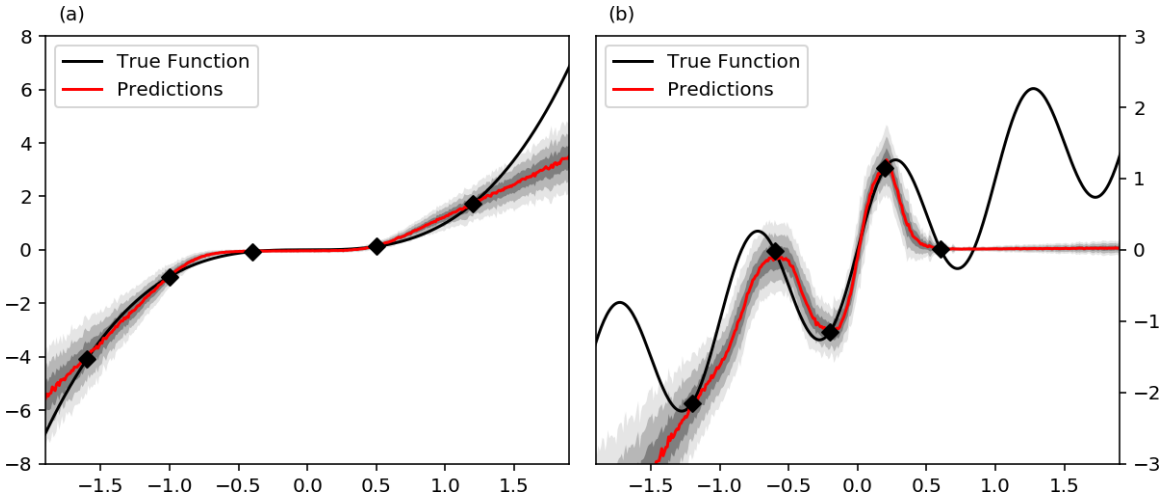

Figure 3.20: This figure shows a single ANN predicting the outputs of learnt functions using the MC-Dropout method. On both figures, the observations are shown by black diamonds, and the uncertainty is indicated by the shaded area. (a) The ground truth function was $y = x^3$. (b) The ground truth functions was $y = sin(x) + x$.

unsurprising, considering we haven't changed the basis for the model nor the training data. The estimated function is smoother when using MC Dropout but this can be attributed to the number of samples taken. For the ensemble method, we trained only 10 networks, whereas we could take 100 predictions using the MC Dropout method with little additional time cost. The main benefit to using MC Dropout instead of an ensemble is that MC Dropout trains much faster; only one ANN needs to be trained. In addition to this, MC Dropout can be applied to any trained DNN, as long as dropout was enabled during the training process. While dropout is a common method for regularisation, many researchers are now using other techniques instead of dropout, such as batch normalisation. Regardless, it is a powerful technique for obtaining uncertainty from an ANN with little additional cost.

3.4.2 Gaussian Processes

While many of the modern successes in ML have come from the use of DNNs, it is only one method of pattern extraction. Another tool that is seeing wide use in ML is GPs, a generic SL method that works effectively on regression problems [157]. As with other SL algorithms, a series of observations with associated ground truths are needed. The GP is then used to construct a model that interpolates between these observations, and form predictions on unseen data. A major distinction of GP models from other SL methods is the ability to incorporate a measure of confidence directly in prediction. As we have seen, obtaining uncertainty with a prediction is a major field of study with DNNs. However, the predictions from a GP model are probabilistic so confidence intervals can be computed.

To understand how a GP model works, first consider the problem of regression where the goal is to learn a function from a training set. With linear regression, fitting a function is a simple task; we assume the function can be modelled with the following equation:

$$f(x) = \theta_1 x + \theta_0 + \epsilon \tag{3.8}$$

where ϵ is an error term used to account for random sampling noise. The task is then to tune θ_1 and θ_0 such that the function best suits the linear relationship of the data. This same strategy can be used as long as the structure of the data is known beforehand. The function is then changed to match the structure, and the number of tunable parameters θ_n will change as a result.

Instead of optimising for a single point estimate for each parameter, we can use Bayesian regression to find a distributions for each of the model parameters. Take the linear regression model, as shown in Equation 3.8. This function can be written in terms of a probabilistic model using a normal distribution with the following

equations:

$$\begin{aligned} \mu &= \theta_1 x + \theta_0 \\ f(x) &\sim \mathcal{N}(\mu, \sigma) \end{aligned} \tag{3.9}$$

The result of performing Bayesian Regression is a distribution of possible model parameters that all model the functions considering the training set. The prediction, therefore, is not estimated from a single input but is drawn from a probability distribution. With a distribution, an average value and confidence intervals can be calculated for the predictions.

When using regression methods such as linear regression, the structure of the model needs to be defined beforehand, and the number of parameters to optimise is fixed. In contrast to this, GPs are a non-parametric Bayesian method that aims to find a probability distribution over all possible functions. Bayesian methods take their name from Bayesian Inference, a method where we start with some belief about the system and update based on information available. Bayesian Inference builds on Bayes' Theorem, mathematically defined as

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{3.10}$$

where A and B are events, $P(A)$ and $P(B)$ are the probabilities of A or B occurring, and $P(B|A)$ is the conditional probability that B occurs given that A has happened. This allows us to use some knowledge or belief that already exists, known as the prior, to help calculate the probability of a similar event. In terms of Bayes' Theorem, the prior event is A, as this event has happened regardless of B.

As with other Bayesian methods, GP methods start with a prior and use the available information to obtain a posterior distribution for the model. Assuming

we want to learn a function f from data $D = \{X, Y\}$, the GP defines the prior distribution over functions, P(f), which can be used for Bayesian regression (Equation. 3.11).

$$P(f|D) = \frac{P(D|f) \times P(f)}{P(D)} \tag{3.11}$$

A GP is a generalisation of a Gaussian distribution [157]. Over a limited domain, any function, f, can be approximated by a vector such as $[f(x_1), f(x_2), ..., f(x_N)]$. A GP is any distribution of functions that ensures any sequential series of function values has a joint Gaussian distribution. A GP is parameterised entirely by a mean function, $u(x)$, and a covariance function, $K(x_i, x_j)$. This covariance function characterises similarity between two points x_i, x_j, and ensures that while these two points are similar, then the output of the function at these points should be similar too. The mean function is traditionally zero everywhere as this information can be encoded into the covariance function.

Kernels

The covariance function, also known as the kernel, defines the generalisation properties of a GP model. The kernel function is a positive definite function that defines the similarity between two values.

$$Cov[f(x_i), f(x_j)] = K(x_i, x_j) \tag{3.12}$$

The kernel specifies the correlation between the function output at two points in the domain of the function, x_i and x_j. Therefore the choice of kernel function specifies what form of functions will be likely in the GP model. As such, for complex problems, expert domain knowledge is required to choose a kernel that is appropriate for the data. Perhaps the most simple kernel uses a linear function (Equation 3.13). Using a

linear kernel in a GP is the same as Bayesian linear regression and will only provide linear functions in the prior. When data is received, the posterior distribution will contain the linear functions with the best fit to the data.

$$K(x_i, x_j) = \sigma_b^2 + \sigma^2(x_i + c)(x_j + c) \tag{3.13}$$

For non-linear GP models, there are many types of kernels which are viable. The choice comes down to what kind of information the user wants to encode in the prior functions. One of the most common kernels used is the Gaussian kernel (Equation 3.14a), also known as the RBF kernel. This kernel accommodates high local variation to be seen in the prior functions and allows for data separated by high distance to be independent. As a result, using an RBF kernel doesn't encourage global structure to emerge. There are other kernels which do give rise to non-local structures. For example, using a periodic kernel (Equation 3.14b) creates a prior of functions that all contain repeating structures.

RBF Kernel

$$K(x_i, x_j) = \sigma^2 exp\Big(- \frac{(x_i - x_j)^2}{2l^2}\Big) \tag{3.14a}$$

Period Kernel

$$K(x_i, x_j) = \sigma^2 exp\Big(- \frac{2}{l^2} sin^2\Big(\frac{\pi(x_i - x_j)}{p}\Big)\Big) \tag{3.14b}$$

Figure 3.21 shows examples of the prior distributions generated using each of the three kernels discussed: linear, RBF, and periodic. We sampled five functions from each prior distribution and visualised them using the coloured lines. The mean function is zero everywhere and shown as a black line. The confidence intervals are also displayed as the shaded areas. The linear kernel is distinguishable because the

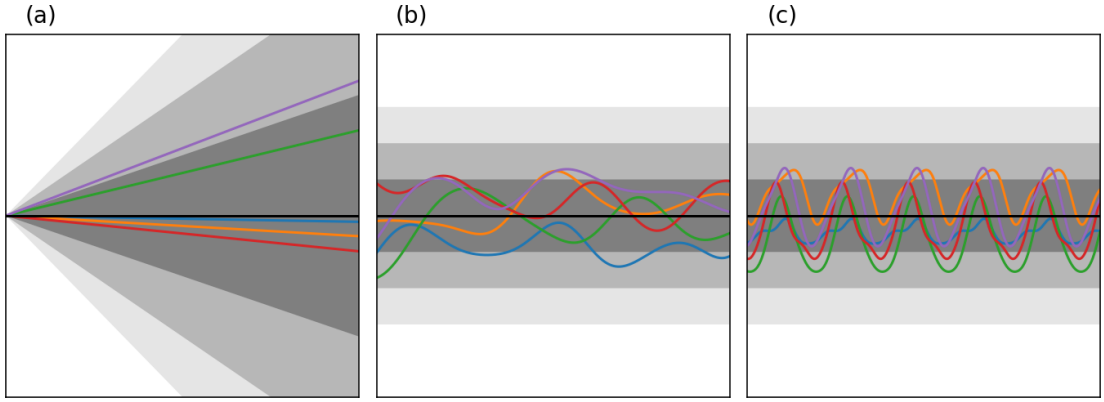

Figure 3.21: Shows the prior distributions for each kernel. We have sampled five functions from each to show the function structures that may occur. (a) Linear kernel function. (b) RBF kernel function. (c) Periodic kernel function.

function depends on the absolute location of the inputs. The RBF and periodic instead only depend on the relative difference between the two inputs; this property is known as stationary. We see that the functions generated using the periodic kernel function have a repeating structure, whereas the RBF generates functions with no global structure.

Making Predictions

Consider the examples visualised in Figures 3.19 and 3.20. In these examples, we had five training points for each function. We denote the training points as X and the function values at these points as f. The test points and outputs are denoted likewise as X_* and f_*. If we were to make a prediction with a GP model using the test datasets, we'd need to incorporate the knowledge from the training set into the model. The prior defines the joint distribution of the function outputs as:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right) \tag{3.15}$$

The posterior function distribution is calculated by restricting the prior function distribution such that only functions that agree with the observed data are valid. Doing so gives the following [157]:

$$p(f_*|f, X, X_*) = \mathcal{N}\Big(\mu(X_*) + K(X_*, X)K(X, X)^{-1}(f - \mu(X)), \\ K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)\Big) \tag{3.16}$$

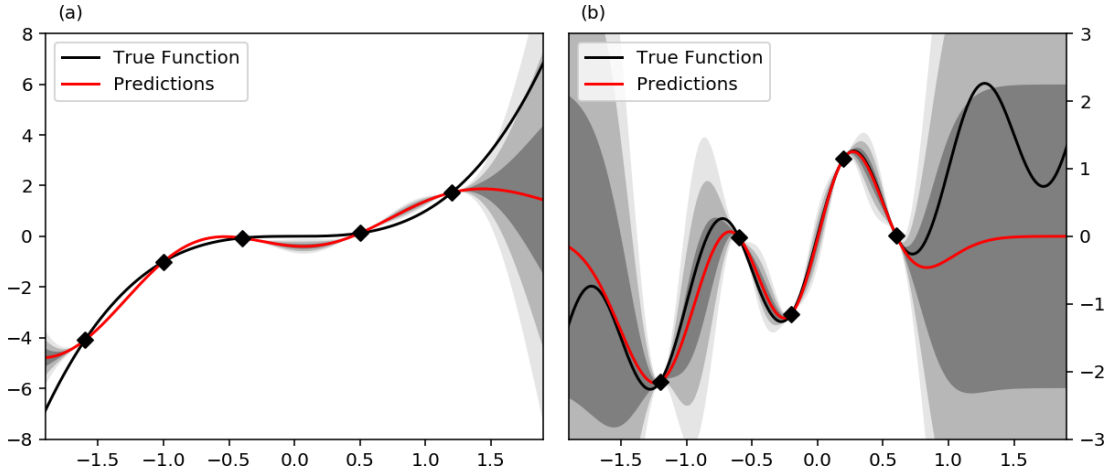

Figure 3.22: Shows the results of a GP model. On both figures, the observations are shown by black diamonds, and the uncertainty is indicated by the shaded area. An RBF kernel was used for both with variance = 1, and lengthscale = 0.3. (a) The ground truth function was $y = x^3$. (b) The ground truth functions was $y = sin(x)+x$.

Figure 3.22 shows the GP model applied to the two functions we saw in Figures 3.19 and 3.20. We see the function has similar performance on observed data as the previous two methods that use DNNs. When the interpolated predictions are not precisely the same as the true function, the correct output is within a single standard deviation range. The extrapolated results are much more improved than the DNN models because the GP model shows high uncertainty on the predictions away from training data. It is most important for our problem that the model will consistently provide a confidence measure, the numerical value of the standard deviation is not

necessarily as important. Unlike the DNN models, we see that the GP model consistently has a high uncertainty away from known measurements. Because of the consistency and the quicker training time of a GP, we will use GPs to create the models for the RL environments. Note that while the computational requirements of a GP scale poorly with the number of observations [157], we will be using a relatively small number of observations, and so we do not expect this to be an issue. To being with, we will use the RBF kernel function as this does not impose global structure on the model.

3.5 Learning Phase Transitions

The goal for this project was to show that an RL agent can learn to create its own model of an environment that it can then act on. With traditional RL methods, the agent interacts directly with the environment. The actions that the agent takes alter the state of the environment in some way and then the agent receives information about the state in return. To allow for an agent to build its own representation, we have altered the traditional RL relationship by introducing a model that the agent interfaces with, hereby referred to as the hypothesis. By adding the hypothesis, we can reduce the number of observations the agent receives from the environment. This would be useful for environments where measurements have a high-sample cost because we can replace many of these measurements with simulation results.

To build the hypothesis, the agent must take specific actions to do so; we refer to these as observations. We differentiate these actions from those that do not return any measurements, referred to as processes. We will be using these terms throughout this thesis, however, in future we will be changing the operation names to align closer with the POMDP formalisation. When an observation is taken, data from the current state is sampled from the environment. The state of the environment is also known as the intrinsic state. Data sampled from the intrinsic state is known to be true within some certainty. In some environments, there can be multiple forms of observations that return different information depending on what measurements that observation samples. The data received from an observation action is passed to the hypothesis, and the model is updated accordingly. A process, on the other hand, does not return any information from the environment; it is an action which is input to both the environment and hypothesis that alters both. If the hypothesis is a perfect model of the environment, then they will respond in the same way. However, if the hypothesis

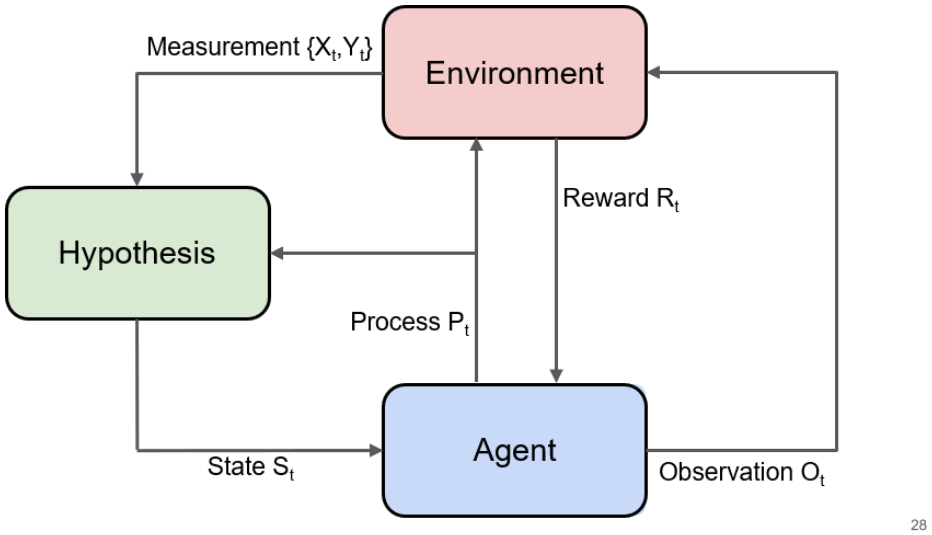

Figure 3.23: Here we illustrate the relationship between the environment, the agent, and the hypothesis.

differs from the environment, then it is possible for the intrinsic state and the state of the hypothesis to diverge. Should this happen, an observation action must be taken to correct the hypothesis.

3.5.1 Playing with Uncertainty

To test this concept, we made an environment that simulates heating and cooling a fixed amount of water by adding or removing energy respectively. The simulation models two characteristics of the water; the temperature of the system and the mass fraction of ice, water, and vapour. This is visualised in Figure 3.24. Neither of these two measures contains complete information about the system in all of the state space. For example, if the system is at $100°C$, adding energy may not show a change in the temperature measurement. In this case, the additional energy will contribute to the phase transition, and so we need an observation of the mass fraction to know exactly what state the system is in. The goal of this scenario was to heat the water to a specific temperature or mass fraction by adding or taking away the right amount of

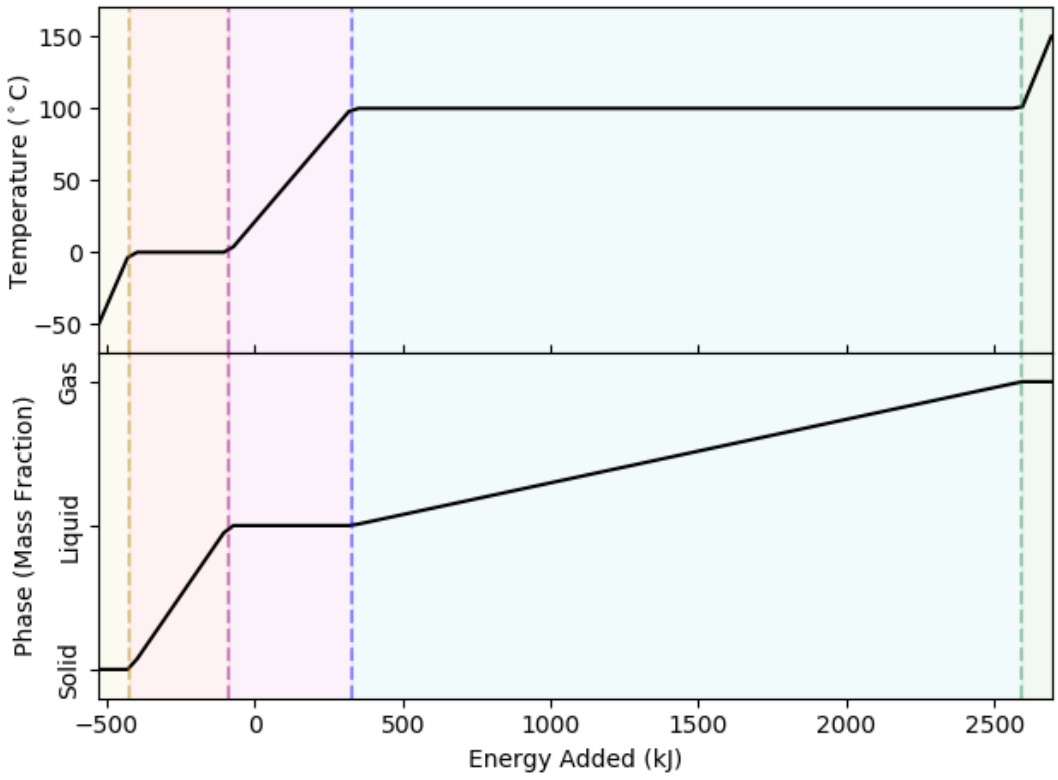

Figure 3.24: Visualisation of the true environment the agent will act in. A single energy value will provide the agent with a temperature value and a mass fraction value.

energy. To do so, the agent needs to learn the state transitions at $0°C$ and $100°C$ with no previous knowledge of the system dynamics. There is also Newtonian cooling to the surrounding temperature as follows:

$$\frac{dQ}{dt} = c \cdot (T(t) - T_{env}) \tag{3.17}$$

The purpose of this environment was to test the ability of an RL agent to build its own hypothesis of the environment. The agent will not receive information from the environment directly. Instead, it uses a local hypothesis that it will have to build as it explores the environment. To create this hypothesis, we used two one-dimensional

GP models; one for the temperature and one for the mass fraction. The data used to build the hypothesis is either $\{E_i, T_i\}$ or $\{E_i, M_i\}$ for the temperature and mass respectively. The agent stores a local memory of both of these datasets that are updated every time a new measurement is taken. The mass fraction output from the environment is a vector of three numbers, shown as $M^* = [m_0, m_1, m_2]$. Given the nature of the phase transitions in this system, it is only possible for two values in M^* to be non-zero. To build a GP model for the mass fraction, we will reduce M^* to a single value using the following:

$$\begin{aligned} I &= \{i : m_i > 0 \; \forall \; i\} \\ M &= min(I) + {M^*}_{min(I)} \end{aligned} \tag{3.18}$$

With this encoding, 0 represents 100% ice, 1 represents 100% water, and 2 represents 100% vapour. Non-integer numbers show there are two phases present, with the decimals indicating the percentage through the phase change. A continuous scale works for this system because the phase changes are sequential. In other words, there is no phase transition between ice and vapour.

The GPs allow for the agent to have an local model of the environment which can be updated when new information is available; we refer to this model as the hypothesis. The agent makes decisions based on this hypothesis, which means that the state passed into the policy is based on the hypothesis. When a prediction is made using the GP model, the agent receives the predicted temperature and associated uncertainty at the agent's current energy from the temperature hypothesis, and likewise the current predicted mass fraction and uncertainty. Using this information, the agent can choose to between four actions; to add or subtract a set amount of energy ($10kJ$), to take a measurement, or to do nothing. When a measurement is taken, new data is collected at the current energy input value, stored with the data

taken up to this point, and the GP hypothesis is recalculated. The section of the reward associated with the temperature of the system at any time is as follows:

$$R_{T_i} = -\Big(|T'_i - T_{target}| + |\hat{T}_i - T_{target}| + \mu_{T_i}\Big) \tag{3.19a}$$

where T' is the true temperature, $\hat{T}$ is the predicted temperature, T_{target} is the target temperature, and μ_{T_i} is the uncertainty in the temperature model. Likewise, for the the mass fraction, the associated reward at any time is as follows:

$$R_{M_i} = -\Big(|M^{*\prime}_i - M^*_{target}| + |\hat{M}^*_i - M^*_{target}| + \mu_{M_i}\Big) \tag{3.19b}$$

where $M^{*\prime}$ is the true mass fraction, $\hat{M}^*$ is the predicted mass fraction, M_{target} is the target mass fraction, and μ_{M_i} is the uncertainty in the mass fraction model. The two are summed to give the total reward. For any one trajectory of a policy in the environment, the assigned score is the reward at the final timestep.

The first goal we set was to train an agent in these conditions and successfully alter the state of the environment such that the water reaches a fixed condition. We used a DNN policy that had two hidden layers, each 128 nodes wide, with the ReLU activation function. To optimise the policy, we used a GA with a population consisting of 100 DNNs. Each generation, the 75 policies with the lowest fitness were replaced. We used the remaining 25 to refill the population with weight mutations that use a learning rate $\sigma = 0.1$. Both of the GP models used the RBF kernel function with lengthscale = 100. The optimisation process was ran five times using a different seed each iteration. The average score during 50 generations of training for three different target values is shown in Figure 3.25. Starting off at room temperature, $21°C$, the agent had to reach a state where 50% of the mass was ice (Figure 3.25a), the temperature of the water was $90°C$ (Figure 3.25b), and where 30% of the mass

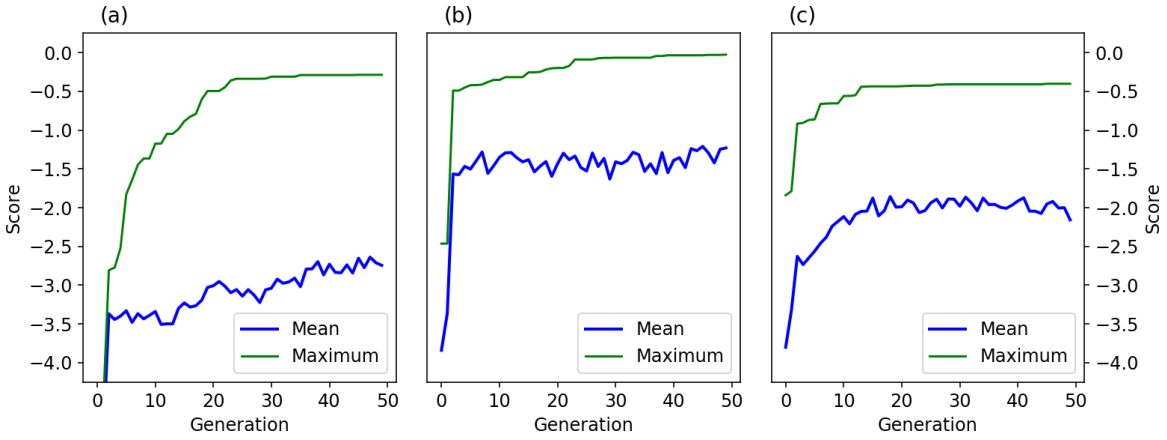

Figure 3.25: Training curves for a policy acting in the water heating environment with fixed targets. (a) The target was to cool the water such that 50% of the mass was ice. (b) The target was to heat the water to $90°C$ (c) The target was to reach a mass fraction of 70% water and 30% steam

was steam (Figure 3.25c). These targets were chosen to show that agent could achieve a range of tasks within the environment.

The optimal score for all of these environments is as close to zero as possible. Such a score would represent the agent perfectly reaching the target, where the hypothesis lines up exactly with the intrinsic state, and there is no uncertainty at the current point. Due to the discrete nature of the steps that the agent can take, it is unlikely to ever achieve a score of exactly zero. However, we do see the best performing polices achieve scores that are very near zero within 50 generations of training in all three environments.

Figures 3.26a and 3.26b show the effect of taking an observation. In 3.26a, the agent, indicated by the blue diamond, is moving into an area where the uncertainty of the hypothesis is getting significant. When the uncertainty is high enough, it will take the observation action, and add a new measurement at the current point (Figure 3.26b). Adding a new measurement will cause the hypothesis to be updated such that the confidence in the current area is higher. This process repeats until the agent has reached its target. In the case of Figure 3.26c, the target was to reach $90°C$.

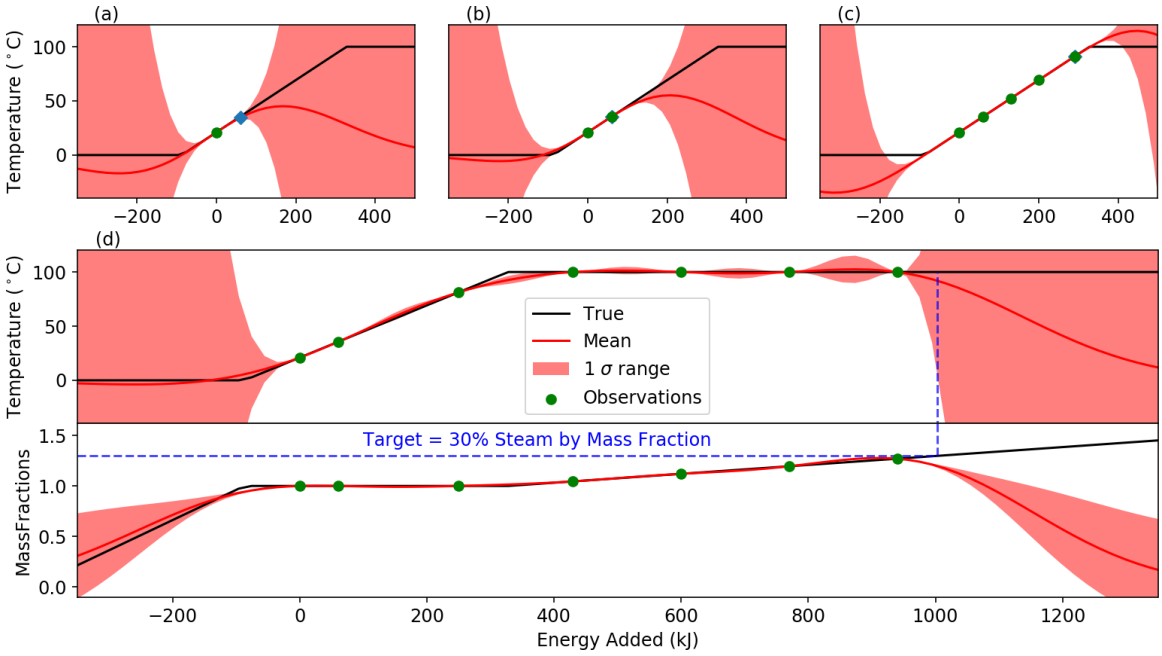

Figure 3.26: The environment after various agents have acted upon it (a) The temperature model a few steps after initialisation. The agent is moving into an area of higher uncertainty that what it has experienced since the start. (b) Shows the new model once an additional observation has been taken. The model now has a higher confidence at the current position of the agent. (c) The agent was tasked to reach $90°C$. Including the initial observation, the agent took five observations to model its path. (d) A separate agent was tasked to reach a state where 30% of the mass is steam. After 50 generations of training, the agent spread its observations out throughout the path it took.

The agent created a hypothesis that covered its path from $21°C$ to $90°C$ with only 5 explicit measurements.

Figure 3.26d shows the final hypothesis for an agent trained to reach a state where 30% of the mass fraction is steam. The observation at the highest energy is not on the target. This is because the agent had 100 timesteps to act in the environment. Notice that the target requires approximately $1000kJ$ of energy to be added from the initial condition, $21°C$. If the agent were to only move towards the target in $10kJ$ intervals, it would still barely reach the target. With taking observations, the agent had even fewer steps to take to get to the final goal. The agent still learnt to the

spread out the observations along its path, and final observation was taken on the final timestep, which increased the reward the agent achieves at the end of the game.

All of the training for these agents was done in simulations. However, should the measurements of the intrinsic state have come from real world data, the final trained agent only requires a handful of these measurements. The process actions did not require a measurement from the intrinsic state, instead only sampling from the hypothesis. This has the effect of reducing the burden on real world-data, and instead using a simulation which has an low sampling cost. To use this algorithm in a real world setting, it would be possible for the agent to be trained entirely in simulation. By doing so, the agent would still learn the need to take observation and build a local model. After training, the agent could then be applied to a real-world setting where it has to explore an unknown scenario and learn the dynamics of the problem.

3.5.2 Expensive Measurements

To optimise our agents, we used a direct policy search algorithm, which alters the ANN policy by randomly fluctuating the network weight values. This means that during training, the observation action may be taken many more times than what we have seen in the trained agents. The driving inspiration for this work was to simulate an RL agent acting in an environment where taking a measurement or observation may be expensive. For example, many chemical experiments require destructive measurements to identify to processes or reactions that have occurred. In such a case, taking many observations will reduce the amount of final product created. Most observations will have some form of cost, even if this is just the time required by the operator. When there is such a cost to observation, that action must be taken intermittently, or it will become unavailable. As a result, taking many observation actions

during training is undesirable. Considering this, we would like to study how an RL agent performs when there are constraints on the number of observations it can take during training.

Therefore, we created a second version of the water environment where we have split the observation action into two separate actions: one to observe the temperature, and the other to observe the mass fraction. Previously, the agent would receive both measurements when the observation action was taken, and so it did not have to account for the different uncertainty in the two GP models. Therefore, we separate the actions to reflect the agent using multiple independent sensors that provide it with alternate information about the environment. The measurements will vary in how informative they are, depending on the current state and so the agent will need to select the most valuable information to sample for the given state. The measurements from the different sensors will each have an associated cost. The agent has a total budget that it can use to take measurements. Once the budget has been used, the agent can no longer gain new information from observations. When acting in these conditions, the agent needs to determine the most valuable information to sample such that it can most efficiently take observations.

Figure 3.27 shows the results for training an agent that could only take six observations in a single iteration of the environment. The environment tracks the number of observations and will not go beyond the set limit. As a result, if the policy requests an observation once the number of observations is at that limit, that action does not change the model or environment in any way, other than occupying a timestep. We used the same training process that was outlined in Section 3.5.1. Comparing to Figure 3.25, we see that the performance is decreased for all training generations, as to be expected when adding additional constraints to the game. Even so, the final score for the agent was close to optimal when the target was to achieve a mass fraction with

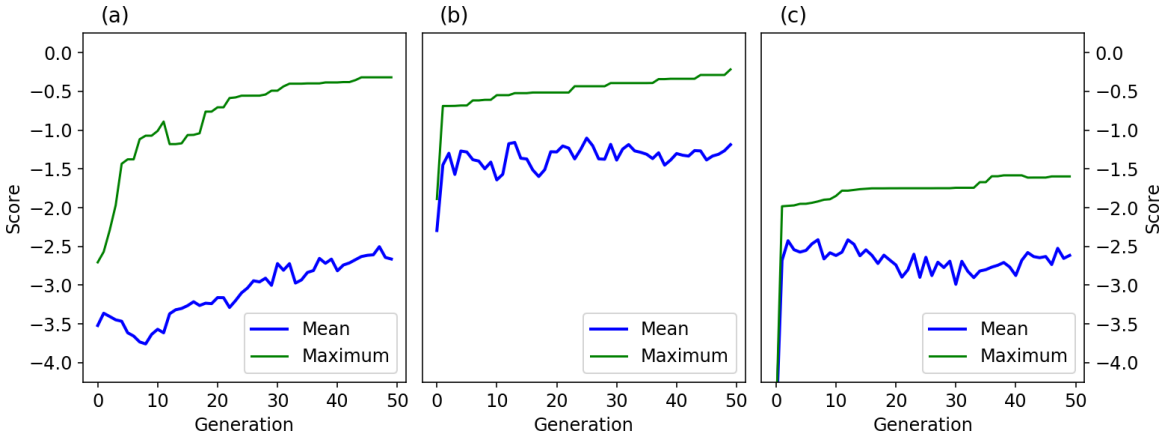

Figure 3.27: Training curves for a policy acting in the water heating environment with fixed targets. The agent could only take six observations during each episode. (a) The target was to cool the water such that 50% of the mass was ice and 50% water. (b) The target was to heat the water to 90°C (c) The target was to reach a mass fraction of 70% water and 30% steam

50% ice and when the target was to heat the water to 90°C. However, the score is noticeably lower when the agent was tasked with achieving 30% of the mass fraction as steam (Figure 3.27c). This is because the amount of energy needed to reach this point is much higher than the other two targets. As a result, the agent needs to take many more observations to build a representative model along the entire path. With the newly implemented budget that limits the number of the observations, the agent can no longer take as many observations as it did, and therefore the score suffers.

To further study the effect of adding a budget, we have examined the case where the agent was tasked with cooling the system to a fixed mass fraction of ice and water. The concept of having a limited budget for actions allows us to alter to cost for the observations to create imbalance. By doing so, we can restrict one form of observation much more than the other for the agent. The agent was given a budget of 30 arbitrary units and could choose how to spend them on the observations. In the following tests, one of the observations actions will cost 10 units, and the other will only cost 1. Doing so forces the agent to adopt a strategy that will sparingly use

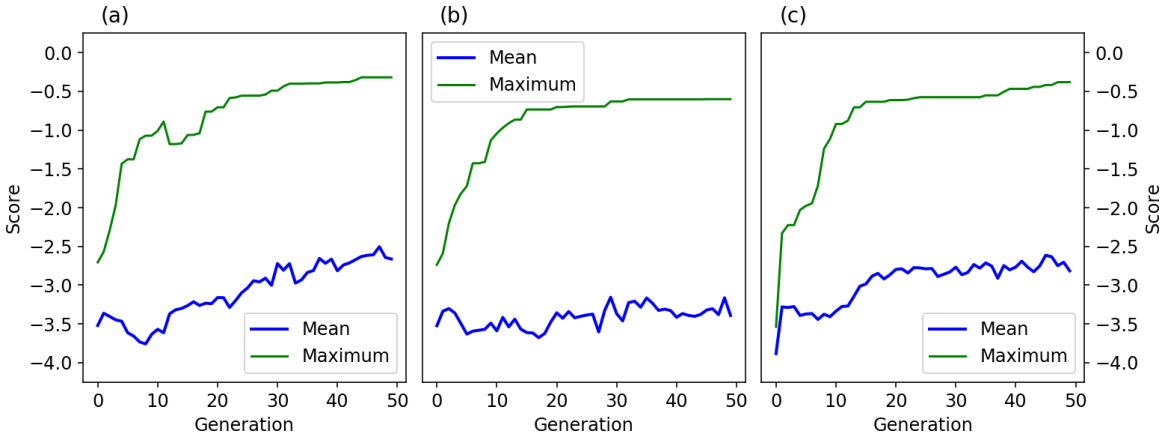

Figure 3.28: Three training curves for a agent that is tasked with cooling the water such that 50% of the mass was ice. Each agent had a budget of 30 arbitrary units. (a) In this iteration both the actions to observe temperature and mass cost 5 units. (b) In this iteration, the cost to observe the temperature was 10, and to observe the mass was 1, (c) The cost to observe the temperature was 1, and to observe the mass was 10.

the expensive observation, but can more freely use the cheaper observation to build one of the models.

Figure 3.29 shows the models obtained from a trained agent. We used the same training procedure for these tests as we did for the previous. For both scenarios, the target was to have 50% of the mass as ice. When the cost for taking a mass fraction observation was high, we see the agent only made one additional mass observation beyond the initialisation, whereas it took several for the temperature model (Figure 3.29a). Likewise, when the cost for taking a temperature measurement was high, the agent would only take two measurements for that model (Figure 3.29b). Comparing the two hypothesis shows them to be very different. When the cost of observing the temperature was high, the agent did not capture the nature of the phase transition well. However, when the temperature cost was low, the agent would surround the target energy with observations to create a model with low uncertainty.

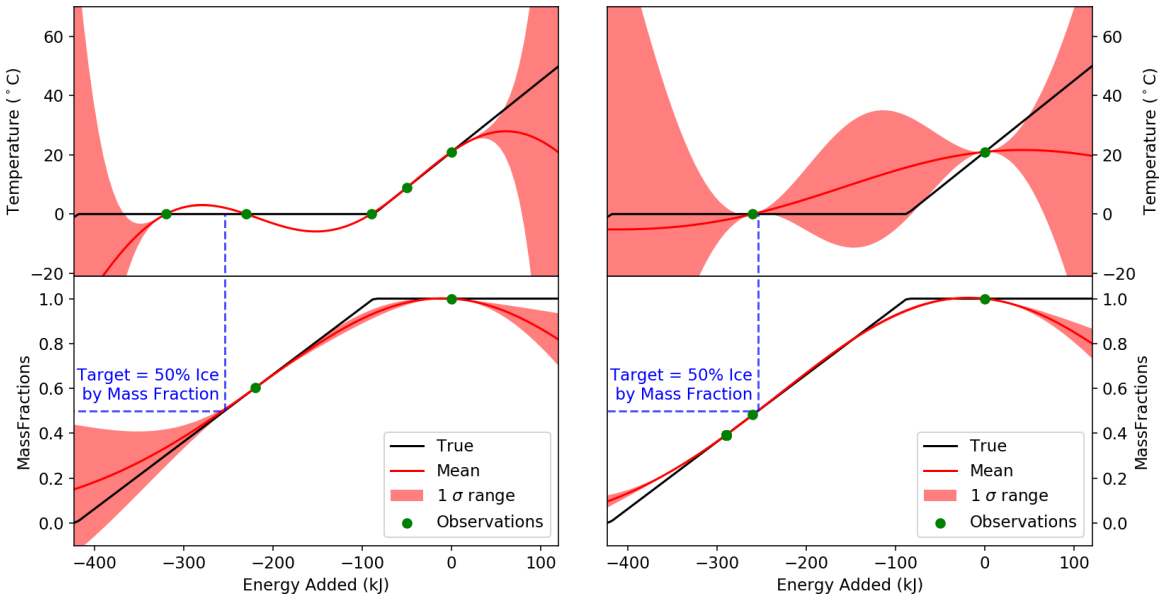

Figure 3.29: The target was to cool the water such that 50% of the mass was ice. (a) The cost to observe the mass is 10, to observe the temperature was 1. (b) The cost to observe the temperature was 10, to observe the mass was 1.

3.5.3 Variable Target

The final version of this environment we are presenting does not have a fixed target, and instead, the target can be varied for a single agent. The agents acting in the previous environments can only be trained to achieve one target. As a result, the usefulness of that policy somewhat limited. We could not use any of those policies to obtain a different target state than what the policy was trained for. To improve the generalisation of the environment, the agent should be able to reach any target in the environment's domain using a single policy. We discussed methods of transfer learning in Section 3.3, however, these methods still require some retraining for each new target.

To achieve a single policy that can alter the state to achieve a variable target, without requiring retraining, we created another version of the environment where the state also included the target temperature and mass values. Every iteration of

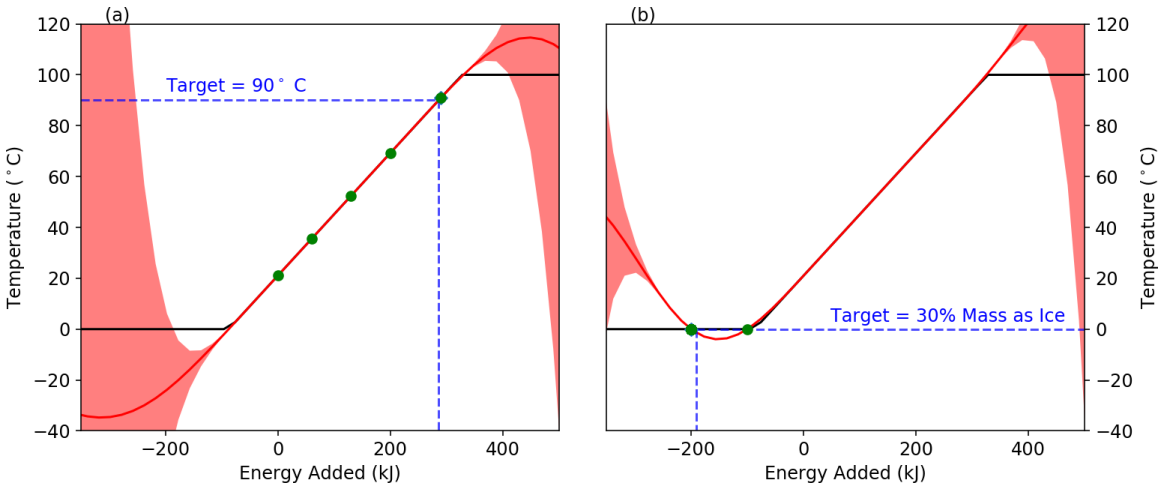

Figure 3.30: (a) The hypothesis of the agent once it had reached the first target of $90°C$ (b) The new hypothesis once the agent had reached the second target, where 10% of the mass was ice. Only the observations take after the the target change are shown.

the environment, a target temperature and mass fraction pair are selected and added to the state, which also includes the information about the current temperature and mass fraction prediction distribution. Additionally, the state can be updated during an episode such that the agent has to change strategy. For RL environments where the goal is to reach a variable area in the state, it is not uncommon for the target to be identifiable in the state. For this test, we used the same training process as described in Section 3.5.1. Figure 3.30 shows a single agent that we trained to reach a variable target that was encoded in the state. The agent was first tasked with heating the water to $90°C$ as we have seen before (Figure 3.30a). After this, we changed the target so that the agent had to cool the water such that the 10% of the mass was ice (Figure 3.30b).

One important result from this agent is that we see it does not take any more observations when in a well-understood area of the state-space. The agent first would take regular observations when heating the water to $90°C$ as seen in Figure 3.30a. However, when the agent was tasked with reaching the second target, it covered

the series of observations that were already taken. Only when it moved beyond the previous measurements did it start taking new observations. This shows that agent is not just taking measurements as part of a repeating pattern, but instead based on an uncertainty dependence.

With this final version of the game, we have shown that an agent can act in an environment with only a handful of observations taken from the true environment. By adding a restriction on the number of observations that can be taken in a single iteration of the environment, the training process can also be done with a limited number of true observations. Importantly, the knowledge learnt in the environment is stable and can be used in future timesteps.

3.5.4 Further Work

There still are some updates we would like to make to this system. Firstly, we would like to enhance the actions available to the agent. For all of the described environments, the agent could add or subtract energy to the system only by explicitly adding $\pm 10kJ$. To give the agent more control of the system, we could make the action to input energy continuous so the agent would have to decide the magnitude as well as the direction. Doing so would allow a well-trained agent to be more precise when reaching the target state.

We would also like to perform further tests on the observation cost. The work done so far with expensive measurement assumes there is a hard limit on the number of observations. Before that limit though, there is no penalty or encouragement such that the agent will take as few observations as possible. Therefore, we would like to investigate alternate methods of incorporating cost into the observation actions. One such method we are considering is to add a penalty to the reward function for every time an observation is taken. Either a fixed value for every observation or perhaps a

variable penalty value that increases every time an observation is taken.

Environments with Higher Complexity

With the new method that we have presented, we have only explored one environment. To further prove that it can be useful, we would need to test the concept on new and different environments. In the presented environment, the agent essentially moves along a one dimensional track; it either increases or decreases the energy value. Given how the environment works, if the agent is moving in the correct direction to reach the target, it it will reach the target in some number of steps.

To make the problem more challenging, we are developing a two-dimensional phase transition environment. This environment is still modelling the transitions between water but will allow for both the temperature and the pressure to be altered. Figure 3.31 shows a phase diagram for water that the new environment models.

Improved Kernel Use

In Section 3.4.2, we discussed how the choice of kernel function defines the prior distribution of available functions from the GP. However, we only discussed a limited number of kernels that were all defined by single functions. Kernel functions can be tailored to reflect expert knowledge of the system, which is especially useful if the desired structure is not given by any standard kernel. To create new kernel functions, we can combine existing kernels together through multiplication and addition [158].

Figure 3.32a shows the result of adding a linear kernel function to a periodic kernel function. We see that adding these two kernel types returns a periodic distribution of functions, but the functions are trending away from the mean. Figure 3.32b shows the result of multiplying a linear kernel function with a periodic kernel function. Multiplying by a linear kernel causes the amplitude of the functions to grow linearly

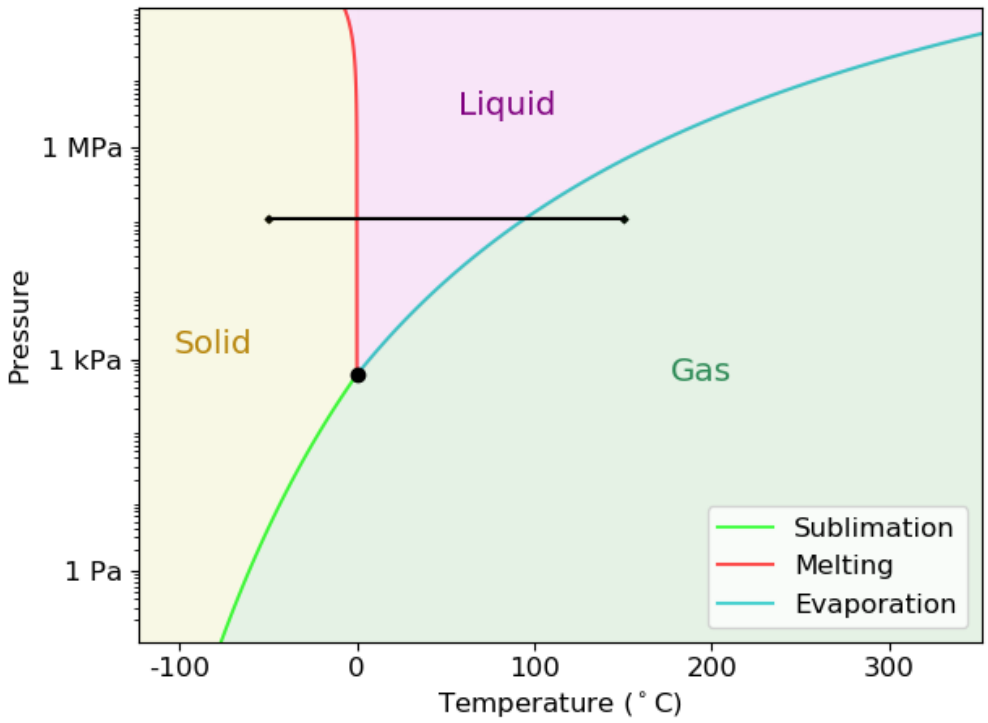

Figure 3.31: A two-dimensional phase diagram for water. We have illustrated the space that the previous environment (3.24) covers with a black line.

away from c, a kernel parameter for a linear function (Equation 3.13).

We would like to investigate how a tailored kernel will alter the model the agent could build in an environment. For the environments presented in this chapter, the RBF function was used because it does not enforce global structure. However, should we want to encode information in the model, we could use a more sophisticated kernel. For example, a linear kernel summed with a RBF kernel (Figure 3.32a) would perhaps capture the increasing trends seen in Figure 3.24.

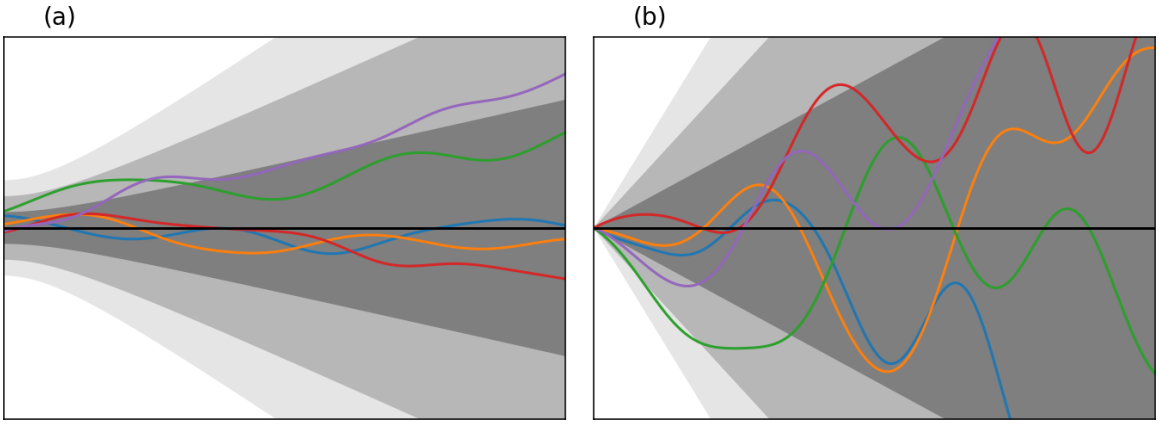

Figure 3.32: Prior distributions of combined kernel functions. (a) The linear and the periodic kernel functions have been summed together. (b) The linear and periodic kernel functions have been multiplied together.

4 Conclusion

In this thesis, we have presented work from two major projects to design tools that utilise ML methods. For the first project, we aimed to create a system that could automatically detect RFI and assist observatory staff in mitigating the interference. In Chapter 2.2, we discussed SL and presented this problem as a computer vision problem, allowing us to use techniques from the literature. The algorithm that was created used a DNN to locate RFI signals in time-frequency space. One important outcome of this project is that we used a DNN that was trained purely on synthetic data, which substantially reduced the resources needed to create a training data set. While more testing needs to be done, we have shown that this method can, at a minimum, be an additional tool to be used when combating RFI at the observatory site. Compared to a traditional RFI detection technique SK, the DNN method was shown to be just as effective at detecting signals with a high signal-to-noise ratio, however the DNN method outperformed the SK test on low signal-to-noise ratio signals. The RFI monitoring project will continue to see updates as the system is established to cover an increased frequency range. The monitor was designed to be scalable without any additional retraining of the DNN, and as a result, the system has been set up to run on a continuous 50Mhz band at DRAO.

An additional aim of this project was to identify the signals which are novel by first characterising signals that are considered normal. To do so, we discussed UL, in

particular, methods of clustering a dataset of discrete points. We have shown that using density clustering methods on the signal data is valid in two dimensions using high-order cumulants. Exploring these clusters in high dimensions will be important when moving forward. We showed that parallel coordinate plots can be used to visually explore the clusters that were identified in two-dimensions in a higher-dimensional space.

The second project focused on developing a RL method that could contribute to automating laboratory experiments. While we did not discuss traditional RL techniques, we instead focused on modern EAs. We showed that these algorithms are a viable method of optimising ANN policies to solve classic RL problems, as well as newly created environments. Additionally, we also showed that training time of well-known environments can be significantly decreased if the policy can be trained in an easier, but similar, environment for just a few generations.

Lastly, we used one of the EAs we discussed to optimise a policy that utilises a Bayesian model to interface with an RL environment. Doing so would allow an RL agent to require less observations directly from an environment, which would important should we be using a physical training environment. We discussed multiple models that could be used when creating the agent's hypothesis of the environment, but decided to use GPs due to their flexibility, ability to incorporate uncertainty directly in the prediction, and low time cost, compared to ANNs. We showed how an agent only needs a few observations to build an accurate model of the domain it acts in. This method allows the agent to reach a target state in the environment with a measurable level of certainty. We also demonstrated how the agent's behaviour changes when certain measurements are limited by cost.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Oriol Vinyals, Igor Babuschkin, et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[4] Kwok-Wai Cheung, James T Kwok, Martin H Law, and Kwok-Ching Tsui. Mining customer product ratings for personalized marketing. *Decision Support Systems*, 35(2):231–243, 2003.

[5] Bimal Viswanath, M Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Towards detecting anomalous user behavior in online social networks. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 223–238, 2014.

[6] Yonghui Wu. Smart compose: Using neural networks to help write emails. https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html, 2018. Online; accessed October 28, 2019.

[7] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):10752, 2019.

[8] Peter Eastman, Jade Shi, Bharath Ramsundar, and Vijay S Pande. Solving the rna design problem with reinforcement learning. *PLoS computational biology*, 14(6):e1006176, 2018.

[9] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[10] Stig Peterson, Meredith Palmer, Ulrich Paquet, and Pushmeet Kohli. Using machine learning to accelerate ecological research. https://deepmind.com/blog/article/using-machine-learning-to-accelerate-ecological-research, 2019. Online; accessed October 28, 2019.

[11] Siyu He, Yin Li, Yu Feng, Shirley Ho, Siamak Ravanbakhsh, Wei Chen, and Barnabás Póczos. Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, page 201821458, 2019.

[12] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431, 2017.

[13] Chris Beeler, Uladzimir Yahorau, Rory Coles, Kyle Mills, Stephen Whitelam, and Isaac Tamblyn. Optimizing thermodynamic trajectories using evolutionary reinforcement learning. *arXiv preprint arXiv:1903.08543*, 2019.

[14] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Zoubin Ghahramani. Unsupervised learning. In *Summer School on Machine Learning*, pages 72–112. Springer, 2003.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[17] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[20] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[21] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[22] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[23] J Bolton, G Stanley, and O Slee. Positions of three discrete sources of galactic radio-frequency radiation. *Nature*, 164:101–102, 1949.

[24] G Reber. Solar radiation at 480 Mc./sec. *Nature*, 158:945, 1946.

[25] L Braes. Radio continuum observations of stellar sources. *Symposium - International Astronomical Union*, 60:377–381, 1974.

[26] G Reber. Cosmic static. *Astrophysical Journal*, 91:621, 1940.

[27] D Edge, J Shakeshaft, W McAdam, J Baldwin, and S Archer. A survey of radio sources at a frequency of 159 Mc/s. *Memoirs of the Royal Astronomical Society*, 68:37, 1959.

[28] D Edge, J Shakeshaft, W McAdam, J Baldwin, and S Archer. Optical identification of 3c 48, 3c 196, and 3c 286 with stellar object. *Astrophysical Journal*, 138:30–56, 1963.

[29] A Hewish, S Bell, J Pilkington, P Scott, and R Collins. Observation of a rapidly pulsating radio source. *Nature*, 217:709–713, 1968.

[30] A Penzias and R Wilson. A measurement of excess antenna temperature at 4080 Mc/s. *Astrophysical Journal*, 142:419–421, 1965.

[31] R Dicke, R Peebles, P Rolle, and D Wilkinson. Cosmic black-body radiation. *Astrophysical Journal*, 142:414–419, 1965.

[32] G Smoot et al. Structure in the COBE differential microwave radiometer first-year map. *Astrophysical Journal*, 396:L1–L5, 1992.

[33] McLaughlin et al. Transient radio bursts from rotating neutron stars. *Nature*, 439:817–820, 2006.

[34] L Spitler et al. A repeating fast radio burst. *Nature*, 531:202–205, 2016.

[35] A Sanna, N Reid, T Dame, K Menten, and A Brunthaler. Mapping spiral structure on the far side of the milky way. *Science*, 358:227–230, 2017.

[36] Umemoto et al. FOREST unbiased galactic plane imaging survey with the nobeyama 45-m telescope (FUGIN) i: Project overview and initial results. *Astronomical Society of Japan*, 69:5, 2017.

[37] The CHIME/FRB Collaboration. Observations of fast radio bursts at frequencies down to 400 megahertz. *Nature*, 566:230–234, 2019.

[38] The CHIME/FRB Collaboration. A second source of repeating fast radio bursts. *Nature*, 566:235–238, 2019.

[39] K Bannister et al. A single fast radio burst localized to a massive galaxy at cosmological distance. *Science*, 365:565–570, 2019.

[40] Industry Canada. Canadian table of frequency allocations. `https://www.ic.gc.ca/eic/site/smt-gst.nsf/eng/h_sf01678.html`. Online; accessed September 15, 2019.

[41] Radio Astronomy Group of the British Astronomical Association. Radio spectrum. `https://www.britastro.org/radio/spectrum.html`. Online; accessed September 15, 2019.

[42] Q Zhang, Y Zheng, S Wilson, JR Fisher, and R Bradley. Combating pulsed radar interference in radio astronomy. *The Astronomical Journal*, 126:1588–1594, 2003.

[43] R Eatough, E Keane, and A Lyne. An interference removal technique for radio pulsar searches. *Monthly Notices of the Royal Astronomical Society*, 395, 2009.

[44] F Meyer, J Nicoll, and A Doulgeris. Correction and characterization of radio frequency interference signatures in l-band synthetic aperture radar data. *IEEE Transactions on Geoscience and Remote Sensing*, 51:4961–4972, 2013.

[45] J Taylor, N Denman, K Bandura, and P Berger. Spectral kurtosis bases RFI mitigation for CHIME. *Journal of Astronomical Instrumentation*, 8, 2019.

[46] D Gary, Z Liu, and G Nita. A wideband spectrometer with RFI detection. *Publications of the Astronomical Society of the Pacific*, 122:560–572, 2007.

[47] G Nita and D Gary. The generalized spectral kurtosis estimator. *Monthly notices of the Royal Astronomical Society*, 406:L60–L64, 2010.

[48] Y Lin, Z Huifang, F Jin, L Ning, and C Jiaqi. Detection and suppression of narrow band RFI for synthetic aperture radar imaging. *Chinese Journal of Aeronautics*, 28:1189–1198, 2015.

[49] G Nita. Spectral kurtosis statistics of transient signals. *Monthly notices of the Royal Astronomical Society*, 458:2530–2540, 2016.

[50] G Nita, D Gary, Z Lui, G Hurford, and S White. Radio frequency interference excision using spectral-domain statistics. *Publications of the Astronomical Society of the Pacific*, 119:805–827, 2007.

[51] M Bailes et al. The UTMOST: A hybrid digital signal processor transforms the molonglo observatory synthesis telescope. *Publications of the Astronomical Society of Australia*, 34, 2017.

[52] K Buch, Y Gupta, S Bhatporia, S Nalawade, and K Naik. Real-time rfi excision for the gmrt wideband correlator. *2016 Radio Frequency Interference (RFI)*, 2016.

[53] The CHIME/FRB Collaboration. The CHIME fast radio burst project: System overview. *Astrophysical Journal*, 863:48–63, 2018.

[54] Y LeCun, C Corted, and C Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`. Online; accessed September 23, 2019.

[55] Y Lecun, L Bottou, Y Bengio, and P Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[56] D Ciregan, U Meier, and J Schmidhuber. Multi-column deep neural networks for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[57] E Kussul and T Baidyk. Improved method of handwritten digit recognition tested on MNIST database. *Image and Vision Computing*, 22:971–981, 2004.

[58] D Keysers, T Deselaers, C Golland, and H Ney. Deformation models for image recognition. *IEEE Transactions on pattern analysis and machine intelligence*, 29:1422–1435, 2007.

[59] B El Kessab, C Daoui, B Bouikhalene, M Fakir, and K Moro. Extraction method of handwritten digit recognition tested on the mnist database. *International Journal of Advanced Science and Technology*, 50:99–110, 2013.

[60] G Cohen, S Afshar, J Tapson, and A Van Schaik. EMNIST: Extending MNIST to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[61] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[62] T Clanuwat, M Bober-Irizar, A Kitamoto, A Lamb, K Yamamote, and D Ha. Deep learning for classical japanese literature. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

[63] D Harrison and D Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5:81–102, 1978.

[64] University of Toronto Delve Development group. The boston housing dataset. `https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. Online; accessed September 23, 2019.

[65] C Cortes and V Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[66] D Decoste and B Schölkopf. Training invariant support vector machine. *Machine Learning*, 46:161–190, 2002.

[67] K Gog, E Change, and K Cheng. SVM binary classifier ensembles for image classification. *Proceedings of the tenth international conference on Information and knowledge management*, pages 395–402, 2001.

[68] Y Lin, F Lv, S Zhu, M Yang, T Cour, and K Yu. Large-scale image classification: Fast feature extraction and SVM training. *Computer Vision and Pattern Recognition 2011*, 2011.

[69] S Huang, N Cai, P Pacheco, S Narandes, Y Wang, and W Xu. Applications of support vector machine (SVM) learning in cancer genomics. *Cancer Genomics Proteomics*, 15:41–51, 2018.

[70] T Subashini, V Ramalingam, and S Palanivel. Breast mass classification based on cytological patterns using rbfnn and svm. *Expert Systems with applications*, 36:5284–5290, 2009.

[71] S Knerr, L Personnaz, and G Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training neural network. *Neurocomputing. NATO ASI Series (Series F: Computer and Systems Sciences)*, 68, 1990. Springer, Berlin, Heidelberg.

[72] H Drucker, C Burges, L Kaufman, A Smola, and V Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems 9, NIPS 1996*, pages 155–161, 1997.

[73] B Schölkopf, J Platt, J Shawe-Taylor, A Smola, and R Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

[74] C Burges. A tutorial on support vector machines for pattern recongnition. *Data Mining and Knowledge Discovery*, 2:121, 1998.

[75] Q Le and M Schuster. A neural network for machine translation, at production scale. `https://ai.googleblog.com/2016/09/`

a-neural-network-for-machine.html, 2016. Online; accessed September 23, 2019.

[76] Barak Turovsky. Found in translation: More accurate, fluent sentences in google translate. https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/, 2016. Online; accessed September 23, 2019.

[77] T Mikolov, K Chen, G Corrado, and J Dean. Efficient estimation of word representations in vector space. *Proceedings of the International Conference on Learning Representations*, 2013.

[78] D Bahdanau, K Cho, and Y Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[79] N Kalchbrenner and P Blunsom. Recurrent continuous translation models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[80] K Cho, B Van Merriënboer, C Gulcehre, D Bahdanau, F Bougares, H Schwenk, and Y Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[81] J Schalkywk. An all-neural on-device speech recognizer. https://ai.googleblog.com/2019/03/an-all-neural-on-device-speech.html, 2019. Online; accessed September 23, 2019.

[82] Young-Bum Kim, Dongchan Kim, Anjishnu Kumar, and Ruhi Sarikaya. Efficient large-scale neural domain classification with personalized attention. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2214–2224, 2018.

[83] Chen L and Y Zhu. Semantic image segmentation with deeplab in tensorflow. https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html, 2018. Online; accessed September 23, 2019.

[84] Zhe Li, Xiaoyu Wang, Xutao Lv, and Tianbao Yang. Sep-nets: Small and effective pattern networks. *arXiv preprint arXiv:1706.03912*, 2017.

[85] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. ACM, 2018.

[86] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016.

[87] Graham Thomas, Rikke Gade, Thomas B Moeslund, Peter Carr, and Adrian Hilton. Computer vision for sports: Current applications and research topics. *Computer Vision and Image Understanding*, 159:3–18, 2017.

[88] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[89] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[90] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[91] K Mills, K Ryczko, I Luchak, A Domurad, C Beeler, and I Tamblyn. Extensive deep neural networks. *Chemical Science*, 10:4129–4140, 2019.

[92] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *SIGKDD explorations*, 4(1):65–75, 2002.

[93] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[94] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[95] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[97] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):19, 2017.

[98] Junhao Gan and Yufei Tao. Dbscan revisited: mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 519–530. ACM, 2015.

[99] scikit learn. sklearn.cluster.dbscan. `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html`. Online; accessed October 20, 2019.

[100] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[101] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10:5, 2015.

[102] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

[103] Vladimir D Orlic and Miroslav L Dukic. Automatic modulation classification: Sixth-order cumulant features as a solution for real-world challenges. In *2012 20th Telecommunications Forum (TELFOR)*, pages 392–399. IEEE, 2012.

[104] MR Mirarab and MA Sobhani. Robust modulation classification for psk/qam/ask using higher-order cumulants. In *2007 6th International Conference on Information, Communications & Signal Processing*, pages 1–4. IEEE, 2007.

[105] Stefan Brecheisen, Hans-Peter Kriegel, Peer Kröger, and Martin Pfeifle. Visually mining through cluster hierarchies. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 400–411. SIAM, 2004.

[106] Alfred Inselberg. *Parallel coordinates*. Springer, 2009.

[107] Christian Bohm, K Railing, H-P Kriegel, and Peer Kroger. Density connected clustering with local subspace preferences. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 27–34. IEEE, 2004.

[108] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[109] Melanie Coggan. Exploration and exploitation in reinforcement learning. *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.

[110] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[111] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[112] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland,

Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[113] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[114] Christopher G Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3557–3564. IEEE, 1997.

[115] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[116] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 2016.

[117] OpenAI. Openai gym beta. `https://openai.com/blog/openai-gym-beta`. Online; accessed September 30, 2019.

[118] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[119] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-

free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[120] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[121] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[122] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[123] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[124] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[125] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[126] X Yu and M Gen. *Introduction to Evolutionary Algorithms*. Springer London Ltd, 2013.

[127] S Risi and J Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9, 2014.

[128] I Rechenberg and M Eigen. *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Stuttgart, 1973.

[129] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[130] J Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3:297–314, 1962.

[131] J Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2:88–105, 1973.

[132] J Holland. Genetic algorithms. *Scientific American*, 267:66–73, 1992.

[133] K Stanley and R Miikkulainen. Evolving neural networks through augmenting topolgies. *Evolutionary Computation*, 10:99–127, 2002.

[134] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, 9(6):653–668, 2005.

[135] Adam Gaier and David Ha. Weight agnostic neural networks. *arXiv preprint arXiv:1906.04358*, 2019.

[136] L Pratt. Discriminability-based transfer between neural networks. *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 204–211, 1992.

[137] Y Bengio, J Louradour, R Collobert, and J Weston. Curriculum learning. *ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48, 2009.

[138] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[139] Luiz A Celiberto Jr, Jackson P Matsuura, Ramón López De Màntaras, and Reinaldo AC Bianchi. Using transfer learning to speed-up reinforcement learning: a cased-based approach. In *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*, pages 55–60. IEEE, 2010.

[140] Manu Sharma, Michael P Holmes, Juan Carlos Santamaría, Arya Irani, Charles Lee Isbell Jr, and Ashwin Ram. Transfer learning in real-time strategy games using hybrid cbr/rl. In *IJCAI*, volume 7, pages 1041–1046, 2007.

[141] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[142] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *IJCAI*, pages 2536–2542, 2017.

[143] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019.

[144] Yuechen Wu, Wei Zhang, and Ke Song. Master-slave curriculum design for reinforcement learning. In *IJCAI*, pages 1523–1529, 2018.

[145] Sanjeevan Ahilan and Peter Dayan. Feudal multi-agent hierarchies for cooperative reinforcement learning. *arXiv preprint arXiv:1901.08492*, 2019.

[146] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[147] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[148] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[149] Marc Peter Deisenroth and Carl Edward Rasmussen. Reducing model bias in reinforcement learning. 2010.

[150] Loic M Roch, Florian Hase, Christoph Kreisbeck, Teresa Tamayo-Mendoza, Lars PE Yunker, Jason E Hein, and Alan Aspuru-Guzik. Chemos: an orchestration software to democratize autonomous discovery. 2018.

[151] Li Shen. End-to-end training for whole image breast cancer diagnosis using an all convolutional design. *arXiv preprint arXiv:1711.05775*, 2017.

[152] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[153] Ugljesa Djuric, Gelareh Zadeh, Kenneth Aldape, and Phedias Diamandis. Precision histology: how deep learning is poised to revitalize histomorphology for personalized cancer care. *NPJ precision oncology*, 1(1):22, 2017.

[154] Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

[155] Sherif Hashem. Optimal linear combinations of neural networks. *Neural networks*, 10(4):599–614, 1997.

[156] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[157] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[158] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.