Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Nicolas Blystad Carbone

Explainable AI for path following with Model Trees

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2020

NTNU
Norwegian University of Science and Technology

Abstract

The increasing amount of applications involving machine learning methods poses a challenge when experts aim to understand their internal workings. Deep Reinforcement Learning (DRL) is a machine learning approach using artificial neural networks to train model-free agents to operate in an environment by trial and error and has shown promising results on many problems. The black box nature of these systems lack transparency in their operation and could lead to lost interest simply because humans cannot trust their decisions. Explainable AI (XAI) aims to provide such transparency and is gaining momentum both in the industry and in academia.

This thesis explores how model trees, in the view of XAI, can be applied and replace a continuous operating agent for path following trained using Deep Deterministic Policy Gradient (DDPG). Model trees with linear and quadratic functions at the leaf nodes are investigated as approximated models of the black box DDPG policy. The Linear Model Tree (LMT) transforms the opaque DDPG policy into a piecewise linear model transparent and traceable from input to output, at the cost of a slightly degraded performance in the environment. LMTs allowed to grow deeper showed improved approximation to the black box policy. The transparency of the tree is reduced proportionally to the depth and a trade-off between accuracy and transparency must be made. The LMT can be used to show each input state's contribution to the consequent LMT action and is usable as an approximated explanation of the state's importance to the DDPG agent's output. The contribution explanations by LMT appear comparable to the explanations provided by the model-agnostic XAI method SHapley Additive exPlanations (SHAP). A series of recommendations are finally proposed for training of model trees when applied to other black box DRL agents. The results are promising and suggests that LMTs may replace black box agents if an expert is satisfied with the behaviour and transparency. As such they deserve further attention in attempts towards new methods in the field of XAI.

Sammendrag

Den økende mengden applikasjoner som involverer maskinlæringsmetoder, utgjør en utfordring når eksperter prøver å forstå deres indre virkemåte. Deep Reinforcement Learning (DRL) er en maskinlæringsmetode som bruker kunstig nevrale nettverk for å trene modellfrie agenter til å operere i et miljø ved prøving og feiling, og har vist lovende resultater på mange problemer. Den sorte boksens natur til disse systemene mangler gjennomsiktighet i virkemåten og kan føre til tapt interesse siden mennesker ikke kan stole på deres beslutninger. Forklarende AI (XAI) har som mål å gi slik gjennomsiktighet og får økende oppmerksomhet både i industrien og i akademia.

Denne avhandlingen utforsker hvordan modelltrær, med tanke på XAI, kan brukes og erstatte en kontinuerlig opererende agent for stifølging trent med Deep Deterministic Policy Gradient (DDPG). Modelltrær med lineære og kvadratiske funksjoner ved løvnodene blir undersøkt som tilnærmede modeller av den sorte boks DDPG-politikk (policy). Lineære Modelltrær (LMT) transformerer den ugjennomsiktige DDPG-politikken til en stykkevis lineær modell som er gjennomsiktig og sporbar fra inngang til utgang, på bekostning av en litt svekket ytelse i miljøet. LMT-er som fikk lov til å vokse dypere, viste forbedret tilnærming til den sorte boks-politikken. Treets gjennomsiktighet reduseres proporsjonalt med dybden, og det må gjøres en avveining mellom nøyaktighet og gjennomsiktighet. LMT kan brukes til å vise hver inngangstilstands bidrag til den påfølgende LMT-handlingen, og kan brukes som en tilnærmet forklaring på tilstandens betydning for DDPG-agentens utgang. Bidragsforklaringene fra LMT virker sammenlignbare med forklaringene gitt av den modell-agnostiske XAI-metoden SHapley Additive exPlanations (SHAP). En rekke anbefalinger blir avslutningsvis foreslått for trening av modelltrær når de brukes på andre sorte boks DRL-agenter. Resultatene er lovende og antyder at LMTs kan erstatte sorte boks agenter hvis en ekspert er fornøyd med oppførselen og gjennomsiktigheten. Som følger fortjener de ytterligere oppmerksomhet i forsøk på nye metoder innen XAI.

Preface

This master's thesis concludes my 10th and final semester at Norwegian University of Science and Technology (NTNU). The work was carried out during the spring of 2020 under the supervision of Associate Professor Anastasios Lekkas. PhD candidate Vilde Gjærum supported the progress through several discussions.

The tools used to perform the experiments includes open source software and multiple libraries. The Python programming language (v3.7.2) was used with the with the NumPy [43], Matplotlib [18], scikit-learn [45] and SciPy [63] packages. The PyTorch deep learning framework [44] is used to build the neural networks in the Deep Deterministic Policy Gradient (DDPG) algorithm. The DDPG algorithm from [28] used in Chapter 4 is based on an implementation by Udacity [60]. The main modifications include a change of noise function to Gaussian distributed noise instead of Ornstein-Uhlenbeck noise and parameter adjustments. The model tree implementation used in Section 2.4 and Chapter 4 is built upon the implementation by Anson Wong [66]. The code was adapted to support polynomial regression at the leaf nodes and rendering of the function expressions when exporting the model tree as a figure. The software package for the model-agnostic XAI method SHAP[32] was used to examine insights in the DDPG policy in Section 4.7.1.

Google Colab (Google Colaboratory) [15], a free Jupyter notebook environment, was used to support GPU-accelerated training of the Deep Reinforcement Learning DDPG agent in Chapter 4. The free file hosting service Google Drive [30] was integrated with Google Colab, enabling the project to be accessible from any computer as long as a stable internet connection was available. Some effort had to be put into solving issues in the Google Colab environment normally not present on a local machine, like saving figures in vector format, calculation timeout, store and retrieve data in Google Drive, but workarounds were found. The version control software provided by GitHub [13] added a layer of security by allowing routinely backup of the code. NTNU provided a workplace at the university with a Dell Inc. OptiPLex 7060 computer and dual monitor setup.

The work behind the thesis relies partly on research performed during the specialisation project fall 2019. Some parts are therefore included while undergoing refinement, modifications and extensions. This includes the exempt on GDPR in Chapter 1, parts on neural networks and optimisation in Section 2.5.1-Section 2.5.4, Requisites for interpretable explanations in Section 3.2, LIME in Section 3.3 and SHAP in Section 3.4 [5]. These are modified and refined such that the thesis can be read independently from the specialisation project. Unless otherwise stated, all work has been performed independently and is original.

Finally, the ongoing COVID-19 pandemic that resulted in the university closing from 12th of March 2020 [55] presented challenges for the work on this thesis. The work was intended to be applied on a drone in collaboration with other ongoing projects. This could've shown whether the proposed method worked in a physical experiment. The scope had to be adjusted accordingly.

Acknowledgements

Thank you to my supervisor Associate Professor Anastasios Lekkas for his valuable insights, suggestions and feedback as the work formed. I would also like to thank PhD candidate Vilde Gjærum for interesting discussions and highly valued comments. Furthermore, thank you to my fellow students and friends who have supported me and allowed for inspiring discussions. Finally, I must express my very profound gratitude to my family for their patience and encouragement throughout this journey.

Nicolas Blystad Carbone,
Trondheim, June 11, 2020

Table of Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence. 1, 2, 4, 23, 62, 67

CART Classification and regression trees. 11

DDPG Deep Deterministic Policy Gradient. i, iii, vi, ix–xi, 4, 5, 22–26, 40–53, 55–64, 67

DRL Deep Reinforcement Learning. i, iii, vi, 3–5, 23, 40, 64, 66–68

LIME Local Interpretable Model-agnostic Explanations. x, 3, 29, 30

LMT Linear Model Tree. i, vi, x, xi, 4, 5, 13, 14, 48–54, 56, 59–68, 79–81

MDP Markov Decision Process. ix, 21

ML Machine Learning. v, 1, 7, 8, 10

NED North-East-Down. x, 35–38, 43

RL Reinforcement Learning. 20, 22

SHAP SHapley Additive exPlanations. i, iii, vi, x, xi, 3, 4, 31–33, 56–60, 62, 67

XAI Explainable Artificial Intelligence. i, iii, ix, 2–4, 27, 28, 68

Chapter 1

Introduction

1.1 Background and motivation

The advancement of Machine Learning (ML) and Artificial Intelligence (AI) systems has enabled computers to achieve impressive results on problems where humans traditionally outperformed machines such as autonomous driving [67, 16], drug discovery [23, 9] and recommender systems [68] amongst others. The advancement in these areas contributes towards an increasingly algorithmic driven society. Though the results are promising, these approaches have been criticised for operating in a black box manner lacking transparency in their operation[52, 1]. This creates difficulties to understand their full workings and ensuring safe behaviour during unexpected events. The opaqueness of modern AI systems could lead to unused potential and lost interest simply because humans can't trust their decisions. For medical applications, where a system reaches an unexpected conclusion, it would be hugely beneficial if the system could explain **why** it reached the prediction so that the doctor and the patient can take well informed decisions. Unknown correlations could be brought to light and guide experts to new theories.

The use of black box automated systems poses safety concerns since operators may not know their limits, hidden biases and flaws. Attention was drawn to this type of use after the two latest real-world accidents with the Boeing 737 MAX Lion Air Flight 610 on October 29, 2018 and Ethiopian Airlines Flight 302 on March 10, 2019. The error that likely lead both flights to crash minutes after takeoff are traced back to a flaw in the automated system MCAS. This was allowed to happen because of a late stage overhaul in the development phase [42]. Pilots were uninformed about the changes in the new planes and were unaware of how the flight control software worked. Because of the improved engines fit on the 737 MAX, the nose would get pushed upwards during takeoff. The MCAS system was supposed to push the nose down such that the aircraft remained in control and avoid a stall during takeoff. This helped the MAX to being similar to the previous version of 737 and consequently avoid airlines spending millions of dollars on further pilot training. As such, most pilots did not know about this system until the crash in October 2018 [42]. An error in the angle-of-attack sensor used by to the MCAS system resulted in the software pushing the nose downwards at unexpected instances and the pilots of Lion Air Flight 610 had to work against the system to keep the aircraft from nose diving [22]. The pilots did not manage to turn off the malfunctioning system in time. Evidence retrieved of Ethiopian Airlines Flight 302 suggests that the aircraft was in a nose dive similar to Lion Air Flight 610 [24]. The cause of accident of Ethiopian Airlines Flight 302 is still under investigation. These two incidents demonstrate the importance of operators needing to know how the black box systems work and ultimately how to turn them off.

As the industry increasingly apply AI methods to support human decision making, it is expected to attain greater responsibility in a transition towards a society driven by automation. The impacts on individuals by these automated decision systems may be significant in cases such as medical treatment, access to loans, credit cards, insurance, employment and so on. The European Union's General Data Protection Regulation (GDPR) is a regulation imposed on all member states of the EU and the European Economic Area (EEA) that went into effect May 2018. It addresses data protection and privacy rights for citizens and aims to give individuals control of their personal data as well as the *right to an explanation* [11]. Recital 71 states:

> The data subject should have the right not to be subject to a decision, which may include a measure, evaluating personal aspects relating to him or her which is based solely on automated processing and which produces legal effects concerning him or her or similarly significantly affects him or her, such as automatic refusal of an online credit application or e-recruiting practices without any human intervention [...]
>
> In any case, such processing should be subject to suitable safeguards, which should include specific information to the data subject and the right to obtain human intervention, to express his or her point of view, to obtain an explanation of the decision reached after such assessment and to challenge the decision.

The exempt expresses that an individual affected by automated decisions have the right to obtain an explanation for a decision and not be subjected to solely automated decisions. These systems therefore need to abide the legislation by providing explanations to individuals who are affected by their decision. This calls for action in the field of AI where not only the importance of insights and security in such systems are relevant, but also their ability to act in accordance to legislation.

The field of Explainable Artificial Intelligence (XAI) has been gaining attention as a response to the rising need for transparent AI systems. It is attempting to open the black box in order to decipher the internal workings, in essence solving the problem of algorithmic opacity. There exist a multitude of methods aimed at guiding experts to decipher these algorithms. A large portion are designed for computer vision tasks and deep neural networks such as Layer-Wise Relevance Propagation [2] and Integrated Gradients [56]. Some methods have been developed to be model agnostic, meaning they can be applied to all black box systems. These could require heavy computation depending on the amount of features to explain. Two commonly mentioned are Local Interpretable Model-agnostic Explanations (LIME) [59] and SHapley Additive exPlanations (SHAP) [33] introduced in Chapter 3 and illustrated in Figure 1.1.

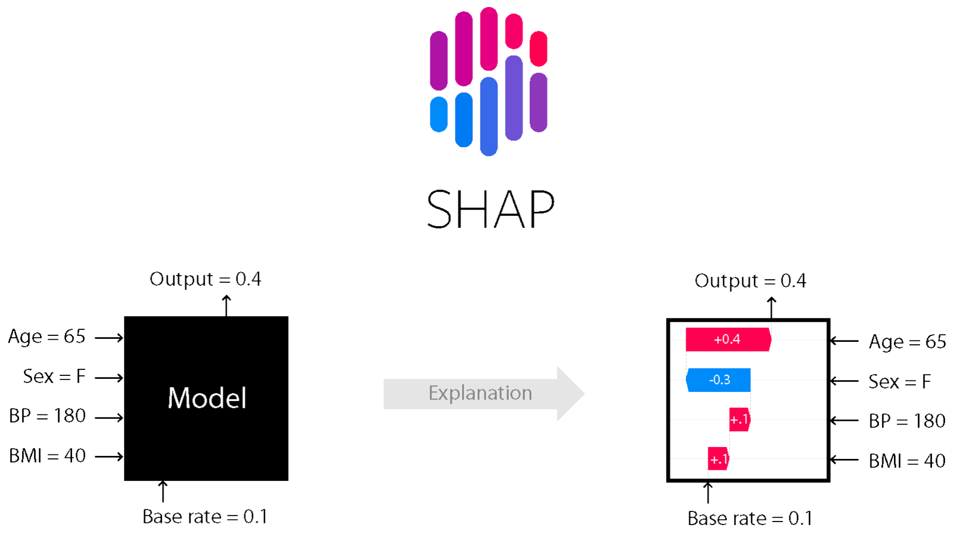


Figure 1.1: An example of an XAI method deciphering the internal workings of a black box model. The importance of each feature in the model is shown as either a positive or negative contribution towards the output. The information can be used to understand how features impact the model prediction. Figure from [32].

Deep Reinforcement Learning (DRL) is the approach of using neural networks and reinforcement learning to enable an agent learning to operate in an environment by trial and error. The behaviour of an agent is known as the policy which suffers from opaqueness when applying current state of the art DRL methods. The combination of DRL and XAI, the scope of this thesis, have seen limited attention in the research community. There are nonetheless some noticeable papers attempting to increase explainability of learned policies of the agents. In [7] the authors succeed to train an agent with Proximal Policy Optimization (PPO) to play Mario AI benchmark with a discrete action space. The DRL-based policy is transformed into a Soft Decision Tree (SDT) of various depths. The SDT uses a neural network as the split condition. Using SDT allowed for valuable insights into the SDT policy, however with a lower performance in terms of episode rewards compared to the DRL-based PPO policy. The authors also discuss the tradeoff between accuracy and interpretability as the SDT became deeper. A drawback noted the by the writers is the lack of explanations in a symbolic form by the SDTs, only in heatmaps and probabilities. It is therefore not clear whether the SDTs may be used to explain the black box PPO policy [7]. In [29], Linear Model U-trees (LMUTs) are proposed to mimic the Q-function of a black box policy. The Q-function uses the state-action pair to estimate the Q-value, the measure for quality of an action in a state. The LMUTs uses a tree structure with linear leaf nodes to approximate the Q-value for each discrete action possible in the given state. The action with the highest Q-value is then selected. By interpreting the approximated

Q-function by LMUTs, the authors extracts rules and may examine the contributions of the input features. It is however not clear how to apply LMUTs for a Q-function in an environment with continuous action agents, if at all possible.

1.2 Objectives

An AI agent learning to control a cyber-physical system using Deep Reinforcement Learning with neural networks will likely discover a nonlinear policy acting as a black box for a human expert. The opaqueness makes these systems difficult to trust and safety could be of concern if they are deployed without a clear understanding of their internal workings. Extensive testing could help, but is not always feasible, nor possible, to test all possible scenarios. This thesis aims to investigate how a black box DRL policy for a cyber-physical system can be transformed into a fully transparent policy though model trees in the view of XAI. Specifically if model trees can be applied on an AI controlled path following system trained using DDPG. It is also of interest to investigate how different depths and leaf node functions may impact both the performance and transparency. Finally, it is aimed to explore if an LMT can provide contribution explanations and how they compare to the model agnostic XAI method of SHAP.

1.3 Outline

This thesis is organised according to the conventional **I**ntroduction,**M**ethods, **R**esults **a**nd **D**iscussion (IMRAD) structure.

- Chapter 1 introduces the reasoning and motivation behind the field of Explainable Artificial Intelligence by looking at scientific and societal impacts of AI. A brief look at previous work in XAI and Deep Reinforcement Learning is presented. The objectives and contributions of the thesis are finally addressed.
- Chapter 2 gives an overview of the theoretic background for the methods used throughout the thesis. This includes ordinary least squares regression, regression and model trees, neural networks, reinforcement learning and Deep Deterministic Policy Gradient.
- Chapter 3 introduces common terminology and two recognised model-agnostic explanation techniques used in XAI.
- Chapter 4 presents the vehicle model and both the straight-line and the curved path following problem. A DRL agent is trained to control the vehicle to follow the straight-line path using DDPG. Next, the black box policy is transformed into a piecewise linear policy using Linear Model Trees of various depths. A piecewise quadratic policy is also created by employing model trees with a quadratic function at the leaf nodes. These are further compared to the original DDPG policy. Using the XAI method SHAP on the DDPG policy is performed in an attempt to provide insights of the black box policy. A discussion of LMT and explainability follows.

Next, the DDPG agent and a suitable LMT agent is tested on the more difficult problem of curved path following before a series of recommendations for transforming a DRL policy into a LMT are presented.

- Chapter 5 concludes the thesis by summarizing the findings and and proposing further work.

Chapter 2

Theoretical background

The presented theory in Section 2.1-Section 2.3 is primarily based on chapter 1, chapter 7 and chapter 16 in the book Machine Learning: A Probabilistic Perspective by Murphy et al. [41]. Note that some of the notation has been slightly altered to increase consistency of the sections. Section 2.4 is primarily based on the original paper of M5-Model Trees [48]. Section 2.5 is based on the recognized Deep Learning textbook by Ian Goodfellow et al. [14] with inputs from the book Artificial Intelligence: A Modern Approach by S.Russel and P.Norvig [51] in Section 2.5.1. Section 2.6 is primarily based on the book Reinforcement Learning: An Introduction by Sutton et al. [57]. Finally, Section 2.7 is based on the work in [28].

2.1 Machine Learning

Machine Learning (ML) is defined in [41] as *a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty*. There are generally considered three main categories in ML, two of them being supervised- and unsupervised learning[41].

The goal in **supervised learning** is to learn a mapping from inputs $\mathbf{x}$ to the the outputs y, given a set of training data. The training data consists of labeled data, meaning input-output pairs $(\mathbf{x}_i, y_i)$ are given. The label, y_i, may be categorical in a finite set (for instance cat or dog) for which the task is called classification. It may also be a scalar, in which the problem is called regression which is discussed in Section 2.2.

When the training set does not have a label, **unsupervised learning** techniques are used to operate on such data. The unsupervised ML methods aims to extract information by finding unknown patterns. This is typically done by clustering data into groups such as in k-means clustering, but other methods like anomaly detection or autoencoders may be appropriate depending on the application.

In the last of the main three categories, **reinforcement learning**, a software agent aims to learn actions in an (un)known environment to maximize its reward signal. The agent acts in trial and error to gather experience in the environment and observes the given rewards. Over time it learns a policy, π, which maps a state s_i to an action a_j by $\pi(s_i) = a_j$. This type of ML is of further introduced in Section 2.6.

2.2 Regression

Regression is a technique used to fit a model to observed targets in a dataset. The fitted model is a function $f : \mathbb{R}^n \to \mathbb{R}$ which later may be applied to predict values given new observations. In its simplest form, regression aims to find the linear relationship in the data. This method is referred to as linear regression and is discussed in Section 2.2.1. Augmenting linear regression to model nonlinear relationships is possible with basis function expansion, further discussed in Section 2.2.2. More advanced regression methods such as regression trees, model trees and artificial neural networks are covered in Section 2.3, Section 2.4 and Section 2.5, respectively.

2.2.1 Linear regression - ordinary least squares

A widely used model for regression is linear regression. The model assumes a linear relationship between the input and output variable and takes the form

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \epsilon = \sum_{j=1}^{N} w_j x_j + \epsilon \tag{2.1}$$

where $\mathbf{w}^\top \mathbf{x}$ is the inner product between the input vector $\mathbf{x}$ and the model weights $\mathbf{w}$. ϵ is the residual error between the predicted linear output and the true value. It is also referred to as the error term, disturbance term, or noise. In ordinary least squares this error is assumed normally distributed, denoted $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$.

A collection of training data $(\mathbf{X}, \mathbf{y})$ is used in order to find the optimal weights $\mathbf{w}$ for the model. The system can be written as $\mathbf{X}\mathbf{w} = \mathbf{y}$ where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1m} \\ 1 & x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}.$$

The first column in $\mathbf{X}$ contain ones such that w_0 corresponds to the intercept term. The system is not guaranteed to have an exact solution and the task becomes to find the coefficients $\mathbf{w}$ that best fits the equation. Residual sum of squares (RSS), also known as sum of squared errors (SSE), is used as the measure of fit in ordinary least squares and is defined as

$$\mathrm{RSS}(\mathbf{w}) \triangleq \sum_{j=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_j)^2. \tag{2.2}$$

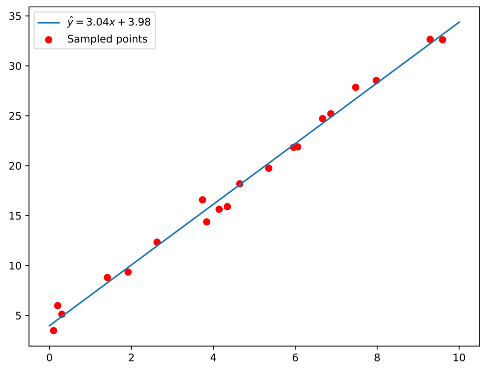
The goal is to find the weights $\hat{\mathbf{w}}$ that minimise the objective function $\mathrm{RSS}(\mathbf{w})$ i.e.

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}). \tag{2.3}$$

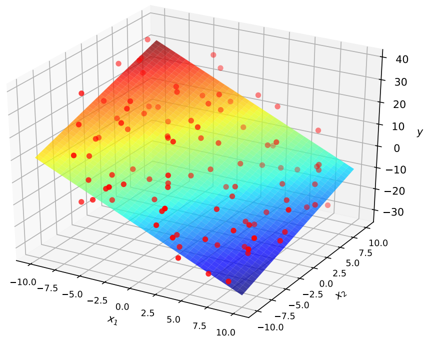
Given that the number of equations n is larger than the m coefficients and that $\mathbf{X}$ is non-singular, then there exist a unique solution for $\hat{\mathbf{w}}$. The gradient with respect to $\mathbf{w}$ in (2.2) is set equal to zero and rearranging gives the expression for $\hat{\mathbf{w}}$ as[1]

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}. \tag{2.4}$$

Two examples showcasing linear regression on noisy data are shown in Figure 2.1.



(a)



(b)

Figure 2.1: Linear regression using residual sum of squares as the measure of fit. a) Regression performed on noisy samples from the function $y(x) = 3x + 4$. The resulting fit is $\hat{y}(x) = 3.04x + 3.98$. b) Regression on noisy 2d data sampled from $y(\mathbf{x}) = -2x_1 + x_2 + 1$. The quadratic fit takes the form $\hat{y}(\mathbf{x}) = -2.03x_1 + 1.04x_2 + 0.33$.

2.2.2 Basis function expansion

Under certain circumstances it is often enough to assume a linear relationship between the input features and the target. However, capturing the underlying nonlinearities may yield higher predictive power and could therefore be desirable. It is possible to model nonlinear relationships using linear regression by replacing the input features $\mathbf{x}$ with a transformation $\mathbf{z} = \phi(\mathbf{x})$. Rewriting (2.1) and applying the transformation yields

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + \epsilon = \mathbf{w}^\top \mathbf{z} + \epsilon \tag{2.5}$$

[1]See Chapter 7.3 in [41] for further details on the derivation.

which is linear with respect to $\mathbf{z}$ and the theory from Section 2.2.1 is again applicable.

As an example, consider a two-dimensional fitted model with a linear regression

$$\hat{y} = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2 \tag{2.6}$$

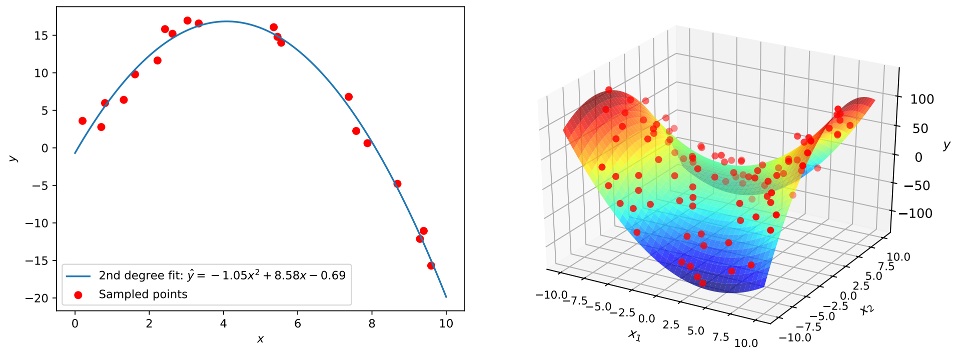
where $\mathbf{x} = [1, x_1, x_2]^\top$. Instead of fitting a plane, a polynomial model can be used. For instance the paraboloid

$$\hat{y}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2 \tag{2.7}$$

suffices in this case. Applying the basis function expansion $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$ to (2.6) yields

$$\hat{y}(\mathbf{z}) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5. \tag{2.8}$$

As a result, the polynomial regression is transformed to linear regression by expanding the model to a higher-dimensional space built by the basis function. Two examples are plotted in Figure 2.2.



(a) **(b)**

Figure 2.2: Linear regression using basis function expansion. a) Regression performed on noisy samples from the function $y(x) = -x^2 + 8x$. The resulting fit is $\hat{y}(x) = -1.05x^2 + 8.58x - 0.69$. b) Regression on noisy 2d data sampled from $y(\mathbf{x}) = 2 + x_1^2 + x_1 x_2 - x_2^2$. The quadratic fit takes the form $\hat{y}(\mathbf{x}) = 0.08 - 0.04x_1 + 0.08x_2 + 1.01x_1^2 + 0.94x_1 x_2 - 0.99x_2^2$.

2.3 Classification and Regression Trees

A classification tree, also referred to as a decision tree, is a rule based prediction method in Machine Learning often described in graphical terms. The predicted variable is categorical, meaning a classification is performed. In a similar way, a regression tree predicts a constant numerical value.

The input space is partitioned into regions with each containing a prediction model [41]. This subdivision may be represented by a tree with branches with a final leaf for each region. The two methods are described as Classification and regression trees (CART) and were coined by Breiman et al. in [4]. Regression trees are mainly discussed forward since the scope of this thesis concerns regression models and not classification.

The model for regression trees can be written on the form

$$f(x) = \mathbb{E}[y|\mathbf{x}] = w_m \mathbb{I}(\mathbf{x} \in R_m) \tag{2.9}$$

where R_m is the m'th region with w_m being the mean value in this region. Classification and regression trees are constructed using the greedy divide-and-conquer method following the procedure in Algorithm 1. There exist different variants of trees, such as CART [4] and ID3 [47], which uses this procedure for growing a tree [41].

Algorithm 1 Recursive procedure to grow a classification/regression tree — *Source: [41]*

```
Require: node, Training data $\mathcal{D}$, depth
 1: function FITTREE($node, \mathcal{D}, depth$)
 2:     node.prediction ← MEAN($y_i : i \in \mathcal{D}$)  ▷ or class label distribution if classification
 3:     $(j^*, t^*, \mathcal{D}_L, \mathcal{D}_R) \leftarrow$ SPLIT($\mathcal{D}$)
 4:     if not WORTHSPLITTING($depth, cost, \mathcal{D}_L, \mathcal{D}_R$) then
 5:         return $node$
 6:     else
 7:         $node.test \leftarrow \{x_{j^*} < t^*\}$
 8:         $node.left \leftarrow$ FITTREE($node, \mathcal{D}_l, depth + 1$)
 9:         $node.right \leftarrow$ FITTREE($node, \mathcal{D}_R, depth + 1$)
10:         return $node$
11:     end if
```

The SPLIT function chooses the best threshold t^* over the set of possible threshold $\mathcal{T}_j$ for feature j

$$(j^*, t^*) = \underset{j \in \{1,\dots,\mathcal{D}\}}{\operatorname{argmin}} \min_{t \in \mathcal{T}_j} \text{cost}(\{x_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{x_i, y_i : x_{ij} > t\}). \tag{2.10}$$

The set $\mathcal{T}_j$ is obtained by identifying each unique value of x_{ij} for feature j. This choice of splitting results in axis parallel splits. The cost function is defined by the expert implementing the regression tree. Common choices are residual sum of squares (RSS), see (2.2), mean squared error (MSE) or mean absolute error (MAE).

The MEAN function is simply the mean over the current regions data, $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$. The function WORTHSPLITTING decides if the cost of the best split from SPLIT satisfies the various possible criterion. Some potential conditions are listed as

- Does the reduction of cost of the split improve the current split? I.e. $\Delta > 0$ where Δ is defined as

$$\Delta \triangleq \text{cost}(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_R) \right) \tag{2.11}$$

- Has the maximum depth not been reached?
- Are the number of samples in both $\mathcal{D}_L$ and $\mathcal{D}_R$ larger than some minimal value?

If all of the selected criterion are True, then the region is split and the algorithm is called recursively in the left- and right child node until WORTHSPLITTING is evaluated to False.

Figure 2.3 shows an example of how the depth of the regression tree affects the prediction accuracy when approximating a fifth degree polynomial. The resulting regression tree depth 5 is further visualised in Figure 2.4.

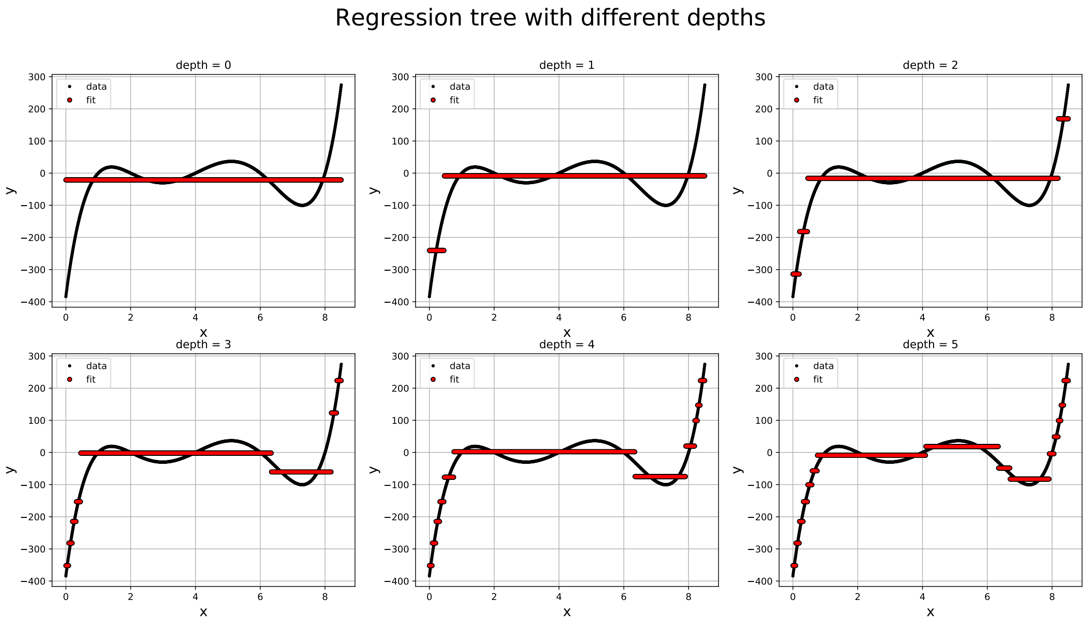


Figure 2.3: The fit of six regression trees trained with 1000 evenly spaced sample points between $x = 0$ and $x = 8.5$ from the continuous function $y = (x - 1)(x - 2)(x - 4)(x - 6)(x - 8)$. Increasing depth results in a closer approximation to the function. The regression tree of depth 5 is visualised in Figure 2.4. Figure made based using [66].

The advantage of CART models are their ability to visually express rules intuitively. They are easy to interpret since the representation is natural for humans. Flowcharts and "How to" manuals are both examples of decision trees used widely in society. The high interpretability of CART models does however come at a price. The most significant drawback is their reduced accuracy compared to other types of models due to the greedy tree construction algorithm [41]. The buildup of a tree also affects the stability; small changes in input may cause significant changes in the tree structure and overall predictions. A common technique to mitigate the high variance in CART models is to train an ensemble of trees, namely random forests [3]. Unfortunately, the simplicity and interpretability of CART models are lost by introducing multiple trees and should be taken into account.

Alternatively, rather than a mean value at the leaf nodes, a linear, polynomial or some other

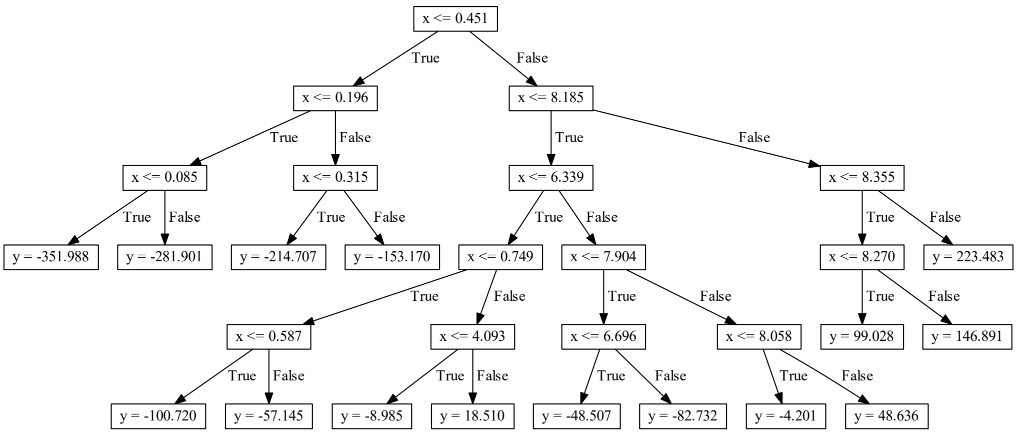


Figure 2.4: The regression tree of depth 5 used in the example in Figure 2.3. The regression tree is read starting from the root node (top). The datapoint, x, is compared to the threshold $x \leq 0.451$. If the condition is True, the left path to the next node is taken. The condition may on the other hand evaluate to False and consequently the right arrow is followed. These steps are repeated until a leaf node is reached and an expression for y is given.

function is possible at the leaf nodes. Using these types of regression at the leaves opens a new type of trees, namely model trees further discussed in Section 2.4.

2.4 Model trees

By introducing multivariate linear regression at the leaf nodes, model trees broadens the concept of regression trees and becomes analogous to piecewise linear functions. However, the function is not constrained to be continuous between the intersection of the input regions. The concept is introduced as M5-Model trees in [48] where the author also suggests regression with other non-linear functions, though at a higher computational cost. The term Linear Model Tree (LMT) is used when referring to model trees with linear regression, while the term model trees describe the general group of trees with various regression models at the leaf node. The main advantage of model trees in comparison to regression trees is the prediction of continuous numerical values in each region instead of a mean value. This reduces the size of the tree and increases accuracy for tasks with very high dimensionality [48].

Building a Model tree

Building a model tree follows the same steps as in algorithm 1, while instead of using a MEAN function, a specified function (i.e. multivariate linear function) is fit using standard regression techniques. For instance the ordinary least squares method introduced in Section 2.2.1. The cost function in M5-Model tree is chosen as the standard deviation [48]. In the implementation used in this thesis it is however chosen as mean squared error (MSE) as in a regression tree introduced in Section 2.3. An example of how a model tree solves the regression task of a fifth order polynomial is shown in Figure 2.5 with the LMT depth 2 visualised in Figure 2.6.

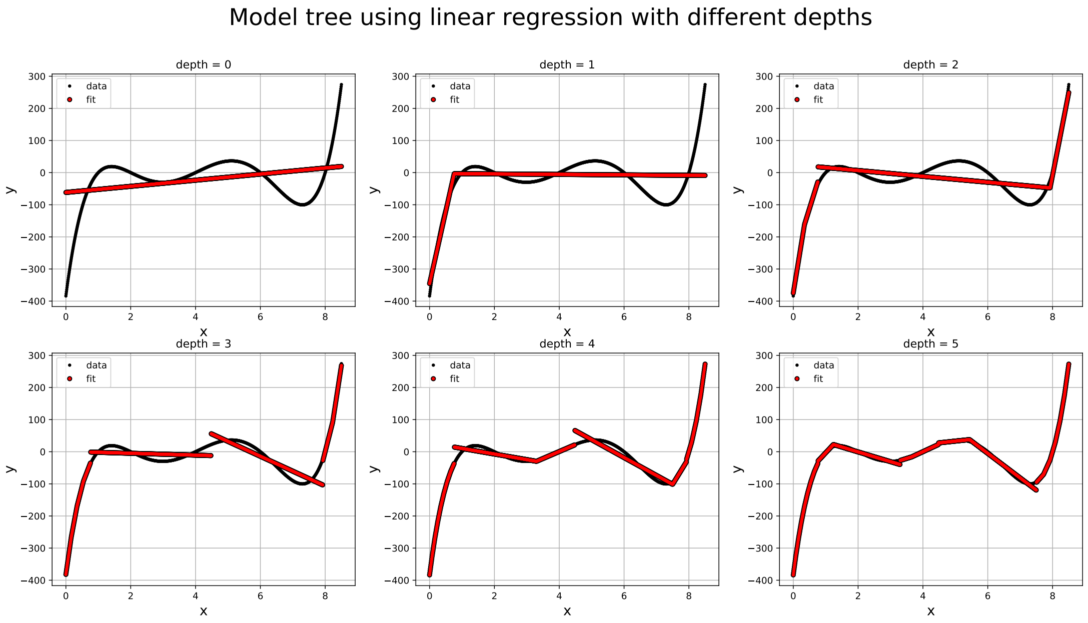


Figure 2.5: The fit of six model trees trained with 1000 evenly spaced sample points between $x = 0$ and $x = 8.5$ from the continuous function $y = (x - 1)(x - 2)(x - 4)(x - 6)(x - 8)$. Increasing depth results in a closer approximation to the function. The model tree of depth 2 is visualised in Figure 2.6. MSE as cost function results in high penalty for points far away from the straight line. Figure made based using [66].

Model trees have several advantages over regression trees by exploiting linear relationships in the data in terms of reduced size while maintaining higher accuracy [48]. The model tree also inherits the intuitive nature of CART models while predicting continuous numerical values. A noteworthy remark is that regression trees always predict a value within the bounds of the training data, while the model tree extrapolates and may predict values outside these (safe) regions [48]. Whether this is an advantage or of concern depends on the nature of the model and application.

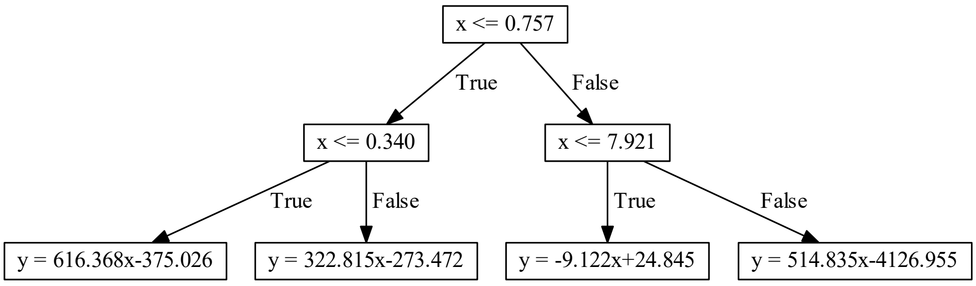


Figure 2.6: The model tree of depth 2 used in the example in Figure 2.5. The regression tree is read starting from the root node (top). The datapoint, x, is compared to the threshold $x \leq 0.757$. If the condition is True, then follow the arrow down along the left path to next node. If the condition on the other hand evaluates to False, follow the path in the opposite direction. These steps are performed until a leaf node is reached and a value for y is consequently given.

2.5 Artificial Neural Networks

Artificial Neural Networks (ANN) can be used as a nonlinear regression method. Before diving into how they may do so in Section 2.5.3, an introduction to the building block of ANN, the Artificial Neuron and some activation functions is provided.

2.5.1 Artificial Neuron

Artificial neurons are the fundamental building blocks in a neural network. They are mathematical models heavily inspired by biological neurons. x_i represents inputs flowing across a link i where each link has a numeric weight, w_i, portraying the strength of the connection. An input bias term b is added and, together with the weighted sum over the n inputs, passed through an activation function $g(\cdot)$. Put simply, each artificial neuron ”fires” or activates based on the linear combination of its inputs. This input to output relation is defined as

$$a = g\left(\sum_{i=0}^{n} w_i x_i + b\right) \tag{2.12}$$

and a visualisation is shown in Figure 2.7. Usually, the x_0 input to a unit is assigned the value 1 with an associated bias weight $w_0 = b$ such that (2.12) can be rewritten as a dot product[2]

$$a_j = g\left(\mathbf{w}^\top \mathbf{x}\right). \tag{2.13}$$

$\mathbf{x}$ is the vector with activations from the previous units with the first element being $x_0 = 1$. The weight vector $\mathbf{w}$ consequently has the bias b as the first element followed by the n

[2]This is sometimes referred to as the *bias trick*. See for instance `http://cs231n.github.io/linear-classify/`

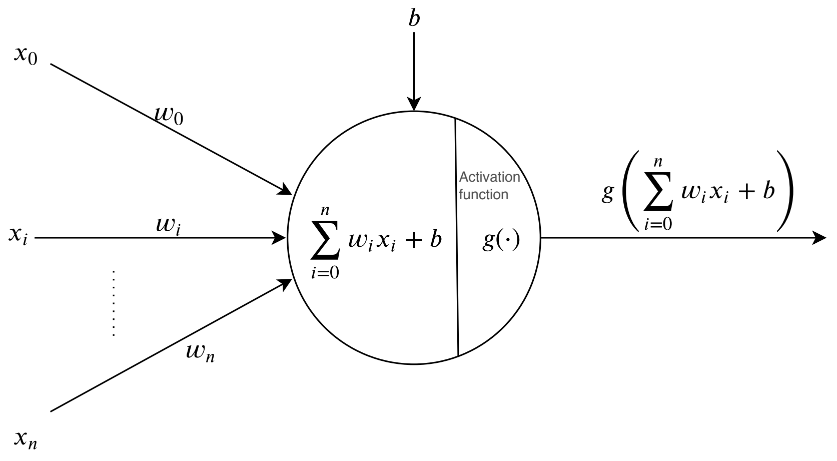

b
x_0
w_0
x_i
w_i
w_n
x_n
$\sum_{i=0}^{n} w_i x_i + b$
Activation function
$g(\cdot)$
$g\left(\sum_{i=0}^{n} w_i x_i + b\right)$

Figure 2.7: One of many possible visualisations of an artificial neuron. The input x_i could be the activation sent from another neuron or a numeric raw value originating from data.

connection weights. This condenses the expression and simplifies the practical implementation by combining the weights and bias into a single vector $\mathbf{w}$. The choice of activation function g determines the amount of activation sent over the link from a neuron j to the next and is chosen to be nonlinear. This is to ensure the important property that the connected network of neurons can represent a nonlinear function.

2.5.2 Activation functions

Sigmoid

Even though there are a vast range of possible activation functions, only a few are commonly used. For starters, the sigmoid is an activation function which constrains the output between 0 and 1 as the input $x \rightarrow \pm\infty$ and this may be seen as a percentage of activation from a neuron. Albeit being simple, it has been heavily criticised for its problem of *vanishing gradients*[17] during backpropagation[3]. The definition of the function is stated as

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.14}$$

with its derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

Relu

Another nonlinear activation function is the rectifier linear unit (ReLu) defined as

$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

[3]Backpropagation and the gradient based learning method is discussed in Section 2.5.4.

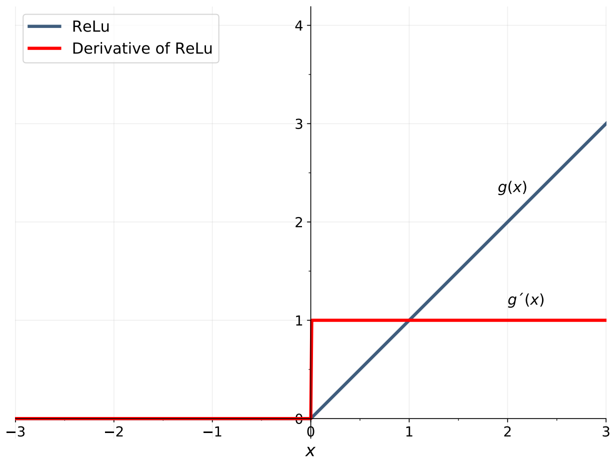


Figure 2.8: The ReLu function plotted with its derivative.

The derivative takes the form

$$g'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{2.16}$$

where it is undefined at $x = 0$. A workaround in practice is to define a value for the derivative at $x = 0$, either the right or left derivative. Unlike the sigmoid, ReLu does not saturate when $x > 0$. If the learned bias b into a neuron is sufficiently negative[4], then the activation from the unit may remain 0 and causes the neuron to never fire. This is essentially the same as removing the neuron from the network and is described as the *dying ReLu problem*[5]. Once this happens, the gradient will forever remain zero as $f'(x) = 0$ when $x < 0$ and no update can correct for the learned parameters into the neuron. An attempt to mitigate this problem is to replace the 0 with αx for $x < 0$ which results in the LeakyReLu function, not further discussed.

Tanh

The tanh activation function squashes its input between -1 and 1 through [6] the function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1. \tag{2.17}$$

[4]This could be a symptom of learning rate set too high or a large gradient
[5]See http://cs231n.github.io/neural-networks-1/
[6]Unlike sigmoid which squashes between 0 and 1.

This activation function also suffers from the vanishing gradient problem and should be avoided for deep neural networks. It can, along with the sigmoid, be used at the final layer to constrain the output value given by the network. This is how it is used in Section 2.7.1.

2.5.3 Deep Neural Networks

In this section Deep Neural Networks (DNN) are referring to what is sometimes called deep feed-forward neural networks or multilayer perceptrons (MLPs). The internal structure is composed by connected layers of artificial neurons. An illustration is presented in Figure 2.9. Each neuron in a layer is connected to every neuron in the next layer and activates based on the signal strength received before it is passed through the activation function, as discussed in section 2.5.1. The first layer can be written as

$$\mathbf{h}^{(1)} = g^{(1)} \left(\mathbf{w}^{(1)\top} \mathbf{x} \right) \tag{2.18}$$

which is passed to the next layer:

$$\mathbf{h}^{(2)} = g^{(2)} \left(\mathbf{w}^{(2)\top} \mathbf{h}^{(\mathbf{1})} \right) \tag{2.19}$$

and so on until the output layer. The biases are included in the weight vector by the bias trick. At the output of the deep network, a final linear or logistic activation function is added depending on whether the network should solve a regression or classification problem. A vector or scalar is used at the output to state the final calculation by the network.

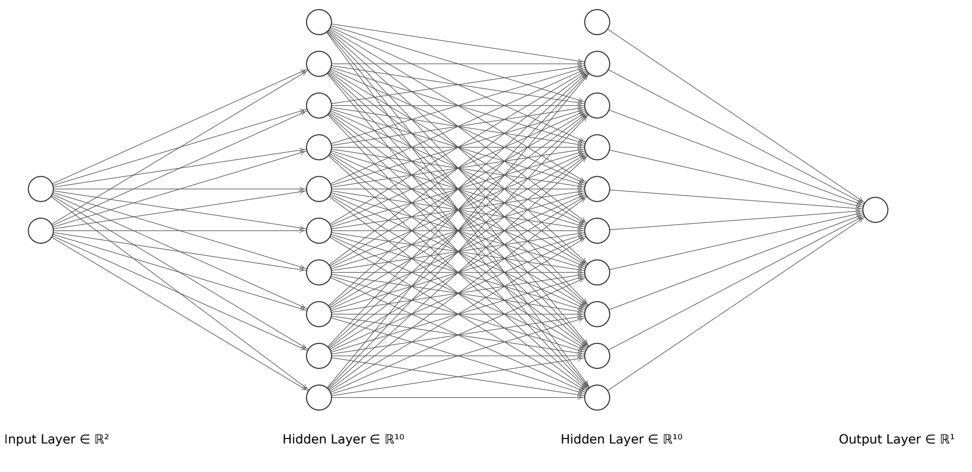


Figure 2.9: A deep neural net with two hidden layers used on a regression problem. The bias term is depicted as a unit activation in the hidden layers. Figure made using the NN-SVG software [26]

The goal in a regression problem is to approximate some function f, such that $y = f(x)$ maps an input x to a value y. The feed-forward network approximates this function by defining a mapping $y = h(x;\theta)$ and finding the parameters θ for the weights through back-propagation, discussed in section 2.5.4. The feedforward term arise because of the direction of computation from input x through the network $h(x;\theta)$ to the output y without any internal feedback connections. The connections between the layers forms a directed acyclic graph.

2.5.4 Loss function and Optimisation

Once the architecture of the deep neural network is defined, tuning of weights and biases are performed. The objective is now to fit a nonlinear function to the input-output space covered by the training data. The training data consist of an input $\mathbf{x}$ with a corresponding label y and is used to tune the network. A cost function, sometimes referred to as a loss function or objective function, measures how the network performs on the training data. The cost function tries to punish the model by a high cost whenever a prediction with a large error is made. Conversely, a low cost is given for predictions with low error and in the ideal case of no error, zero cost is attained. There are multitudes of possible cost functions, a common is Residual sum of squares (RSS):

$$L(\mathbf{x}, y, \theta) = \text{RSS}_{\text{train}} = |\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}|^2 \tag{2.20}$$

where the loss increases squared with the Euclidian distance between prediction and the target becomes larger. This means that large errors are heavily punished by the loss function. Other used loss functions include Mean Absolute Error Loss (MAE) and Mean Squared Logarithmic Error Loss (MSLE) which does not punish large error as much as MSE. This could be beneficial whenever the regression problem consist of outliers far from the mean.

The loss function now quantifies a measure of the performance of the network. Whenever the loss is high, the network performs poorly, while a low loss signifies more accurate predictions. This property is used in the training by changing the weights in the layers of the network through optimisation by minimising the loss. The nonlinearities in a neural network causes the loss functions of interest to be non-convex[7], thereby ruling out useful properties obtained in convex optimisation. Therefore, iterative, gradient-based optimisers are used to minimise the loss function. Unfortunately, because of the nature of nonlinear programming, the minimum is unlikely to be a global minimum, no convergence is guaranteed and the minimum is sensitive to the initial parameters. This means that the weights and biases in a feedforward neural network impacts the resulting minimum found through optimisation. The optimisation procedure by a gradient-based optimiser, gradient descent, uses the gradient of the loss function to change the weights, θ, such that the loss decreases. The weights are updated according to the gradient descent algorithm

$$\theta \leftarrow \theta - \eta \mathbf{g} \tag{2.21}$$

where η is the learning rate. The loss is calculated over all training samples, but may also be updated more than once during a training iteration. This is referred to as updating the

[7]Chapter 6.2 in Deep Learning book[14]

weights using *minibatches*. A *minibatch* contains a smaller set of samples, drawn uniformly from the training set, and is useful if the computer struggles to keep all information in its memory. These are drawn until the complete training set has been used. Whenever gradient descent is used with *minbatches*, it takes the name Stochastic gradient descent (SGD). The gradient $\mathbf{g}$ in (2.21) is found by computing

$$\mathbf{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta) \tag{2.22}$$

where L is a chosen loss function. The (Stochastic) gradient descent algorithm is one of many possible optimisation algorithms and the book [14] gives a detailed introduction to the most common ones, such as SGD with momentum, SGD with adaptive learning rates (Adam, AdaGrad, RMSProp) and Newton's method.

2.6 Reinforcement Learning

The theory presented in this section is primarily based on the book Reinforcement Learning: An Introduction [57].
Reinforcement Learning (RL) is defined in [20] as the problem faced by an agent that learns behavior through trial and error interactions with an environment. The behaviour performed in each state by an agent is called the policy. There are generally two strategies for solving reinforcement learning problems [20], the first being to search for a behaviour performing well in the environment. This is the strategy of evolutionary algorithms like genetic algorithm, genetic programming and evolutionary programming. These algorithms are inspired by the way biological evolution produces skilled behaviour even though learning is not performed during its lifetime. Evolutionary algorithms starts by creating multiple agents with random behaviour and observes which operates best according to a fitness function. New agents are created by crossover breeding and mutation from the top performing agents. The least fit agents are discarded in this step. The new agents are released in the environment and a new cycle of crossover breeding and mutation is again performed on the best fit agents until a satisfactory result is obtained. The method is effective when the policy space is sufficiently small or a lot of time is available for the search [57]. It also have the advantages in partially observable environments.

The second strategy, which is the focus in this thesis, observes and learns by interacting with the environment. These methods can be much more efficient than evolutionary algorithms since they utilise statistical techniques and dynamic programming to estimate the optimal actions. Evolutionary algorithms do not take advantage of the fact that a policy can be a function based on the current state. It also discards information about earlier visited state-actions pairs which could improve the search for a policy. In rare instances could the observed data be misleading if the information is misperceived or incorrect, but this is often not the case for many problems.

In Reinforcement Learning (RL) an agent is assumed to interact with a learning environment E at discrete timesteps t. The environment can be stochastic and is usually modelled

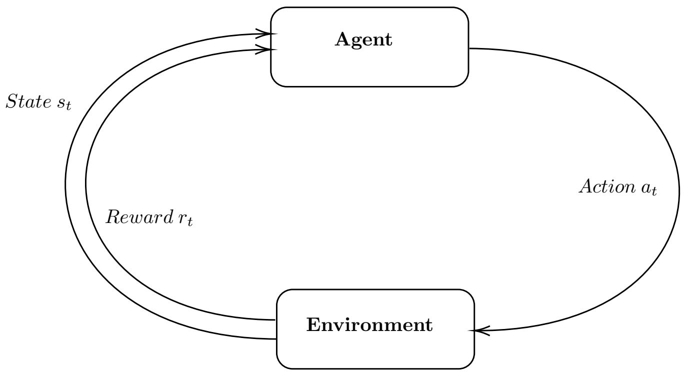


Figure 2.10: The interaction between the agent and the environment in a Markov Decision Process. Figure inspired by [57].

as a Markov Decision Process (MDP). The agent observes the current state $s_t \in S$, performs an action $a_t \in A$ and receives a reward $r_t \in \mathbb{R}$.

2.6.1 Markov Decision Process

A Markov Decision Process (MDP) is a sequential decision problem with a fully observable stochastic environment using a stochastic transition model and additive rewards. The MDP consist of states, s, actions a and a transition model $p(s_{t+1}|s_t, a_t)$ describing the probability of ending up in a state given a state-action pair[51]. In short, an agent observes the current state s_t, performs an action a_t and receives a reward r_t as illustrated in Figure 2.10. Since the final end state may be far away, the agent needs to achieve balance between immediate and delayed rewards.

Reward

The reward given to the agent is a signal used to guide or punish the agent based on its performed actions and current state. The agent seeks to maximise the cumulative reward over the whole episode and therefore needs to trade off immediate and delayed rewards. The rewards may be given to the agent dependant on the objective. For instance, a balancing robot could be given a +1 reward for each second it stays alive and it is therefore encourage to remain in balance. If the goal is to escape from a maze, then a -1 reward may be given each second to promote a faster response to reach the end of the maze and finish the episode. No matter the type of environment, the agent in a MDP always aims to maximise its cumulative reward. It is therefore important to set up the rewards such that they indicate what the programmer wants to achieve.

Return

To enforce the agent to tradeoff between immediate and future rewards, the concept of discounted rewards is introduced. The agent now seeks to maximise the cumulative discounted rewards it receives by

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.23}$$

where $\gamma \in [0, 1]$ is the discount rate deciding the importance of later received rewards. If $\gamma = 0$ then the agent seeks to maximise the reward received in the next step, while a $\gamma = 1$ weights the distant rewards as highly as immediate. In order to ensure the expected discounted return in (2.23) to be bounded, then $\gamma < 1$. As γ approaches 1, the future rewards are weighted more and influences the agent strategy. The policy, denoted $\pi(s)$, recommends an action for each state and is the strategy which the agent follows towards its goal state. It may change over time due to exploring and finding better strategies in the environment. Each policy is measured by the expected utility of the policy[51] where the policy with the highest expected utility is called the optimal policy and is expressed $\pi(s)*$.

2.6.2 Policy function and Value function

Value functions describe the value of being in a state or performing an action in a state such that the agent may estimate the expected return from that state or action. It can therefore be used to select the actions which maximizes the expected return. The rewards given to an agent is dependant on the actions taken (and consequent states visited). The value function is therefore defined in terms of the possible action strategy, namely the policy [57].

The state-value function under policy π is the expected return when starting in state s and following the policy π thereafter. This is written as

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right], \text{ for all } s \in S \tag{2.24}$$

The value of performing the action a in state s and follow policy π can be expressed in a similar way as

$$q_\pi(s, a) \triangleq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a\right] \tag{2.25}$$

which is referred to as the action-value function. These two functions can be estimated by interacting with the environment and record the rewards obtained under the policy π. If the amount of states and actions are infeasible large (for instance continuous), then v_π and q_π may be parameterized. This done in DDPG further covered in Section 2.7.1.

Optimal policy

The overall task in a RL task is to find the policy that achieves the highest expected reward. A policy is better than another policy if the properties of $\pi \geq \pi'$ if and only if $v_\pi \geq v_{\pi'}$ holds.The policy that is better than or equal to all other policies is denoted the optimal policy, π^*. The optimal state-value function is consequently defined as

$$v^*(s) \triangleq \max_{\pi} v_\pi(s), \text{ for all } s \in S \tag{2.26}$$

Similarly, the optimal policy may be written in terms of the optimal action-value function as

$$q^*(s,a) \triangleq \max_{\pi} q_\pi(s,a) \tag{2.27}$$

In the end, the optimal policy describes the solution to which strategy an agent should perform in the environment. As such, the optimal value function v^* can be expressed as

$$\begin{aligned}
v^*(s) &= \max_{a \in A} q_\pi(s,a) && (2.28)\\
&= \max_a \mathbb{E}_{\pi^*}\left[G_t | S_t = s, A_t = a\right] && (2.29)\\
&= \max_a \mathbb{E}_{\pi^*}\left[r_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] && (2.30)\\
&= \max_a \mathbb{E}_{\pi^*}\left[r_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a\right] && (2.31)\\
&= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v^*(s')]. && (2.32)
\end{aligned}$$

This is known as the Bellman optimality equation for v^*. For $q^*(s,a)$ this is written as

$$\begin{aligned}
q^*(s,a) &= \max_{\pi} q_\pi(s,a) && (2.33)\\
&= \mathbb{E}\left[r_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a\right] && (2.34)\\
&= \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') | S_t = s, A_t = a\right] && (2.35)\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q^*(s',a')\right] && (2.36)
\end{aligned}$$

With these expressions, it is possible to find the $v^*(s)$ for each of the states in the environment by solving the set of equations. This can similarly be performed for q^*. Once $v^*(s)$ is found for all states, an agent simply chooses the neighbouring state with highest $v^*(s)$. If q^* is used instead, then the agent chooses the action that maximizes $q^*(s,a)$ in the current state, i.e. $a^*(s) = \text{argmax}_a\, q^*(s,a)$.

2.7 Deep reinforcement learning

Deep Reinforcement Learning (DRL) uses neural networks as function approximations in a reinforcement learning setting. It enables reinforcement learning to train end-to-end, for instance to play video games by receiving only the pixels and score[39, 28].

2.7.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG), introduced in [28], is a model-free off-policy algorithm categorised under the AI branch of reinforcement learning, specifically Deep

Reinforcement Learning. The algorithm consists of two deep neural networks, one to propose a continuous action (actor network) and another network to evaluate the state-action pair (critic). Their relationship is illustrated in Figure 2.11. The actor with network weights θ^μ receives the state s as input and outputs an action a based on the deterministic policy $\mu(s)$. The critic with parameters θ^Q takes both the state s and the actor's action a and gives out the Q-value for this state-action pair. The use of a neural network to calculate the Q-value instead of a table makes it possible for the DDPG algorithm to operate on continuous states and actions. It is a continuation of the previous work on deterministic policy gradient [54] which establishes the underlying theory.

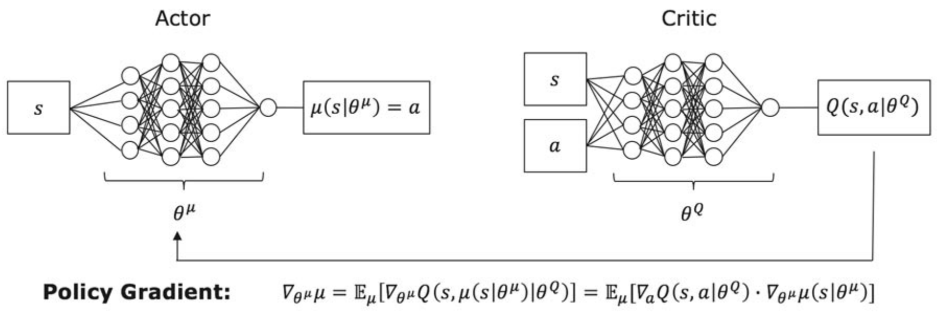

Actor
Critic
s
$\mu(s|\theta^\mu) = a$
s
a
$Q(s,a|\theta^Q)$
θ^μ
θ^Q
Policy Gradient:
$\nabla_{\theta^\mu}\mu = \mathbb{E}_\mu[\nabla_{\theta^\mu}Q(s,\mu(s|\theta^\mu)|\theta^Q)] = \mathbb{E}_\mu[\nabla_a Q(s,a|\theta^Q)\cdot\nabla_{\theta^\mu}\mu(s|\theta^\mu)]$

Figure 2.11: The structure of the actor-critic networks in the DDPG algorithm. The weights of the actor and critic are updated based on the received reward according to Algorithm 2. Figure from [27].

DDPG bases itself on using the deterministic version of the Bellman equation which is stated as

$$\begin{aligned} Q^\mu(s_t, a_t) &= \mathbb{E}[G_t|s_t, a_t] && (2.37)\\ &= \mathbb{E}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]. && (2.38) \end{aligned}$$

This is used to learn the parameters for the critic, θ^Q, by minimising the loss

$$L(\theta^Q) = \mathbb{E}\left[(Q(s_t, a_t|\theta^Q) - y_t)^2\right] \tag{2.39}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^Q). \tag{2.40}$$

The parameters for the critic are then updated using gradient ascent by

$$\theta^Q \leftarrow \theta^Q + \alpha^Q \nabla_{\theta^Q} L(\theta^Q) \tag{2.41}$$

Since the action space no longer is discrete, the equation to find the optimal action,

$$a^*(s) = \underset{a}{\operatorname{argmax}}\, q^*(s, a), \tag{2.42}$$

becomes infeasible to calculate. This function is also needed to be run every time the agent wants to perform an action. Since the action space is continuous, DDPG takes advantage of $Q(s_t, a_t)$ being differentiable with respect to the action, a. This allows for a gradient based learning rule to find the policy, $\mu(s|\theta^\mu)$. The actor uses the gradient of the expected return ($Q(s_t, a_t)$) which is stated as

$$\nabla_{\theta^\mu} J \approx \mathbb{E}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \tag{2.43}$$

$$= \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_t\right] \tag{2.44}$$

and updates the actor using gradient ascent according to

$$\theta^\mu \leftarrow \theta^\mu + \alpha^\mu \nabla_{\theta^\mu} J \tag{2.45}$$

Since the DDPG algorithm aims to mimic the Q-function using neural networks, it is unstable during training if updated directly [28]. Two additional networks are therefore used to provide soft updates and stabilise the training. These networks are known as target networks, Q' and μ'. The soft update follows the scheme

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

where $\tau \ll 1$ is the update rate.

A replay buffer R of finite size is employed during the training phase of the DDPG. Transitions sampled from the environment, (s_t, a_t, r_t, s_{t+1}), are stored in this buffer where the oldest sample is removed whenever it becomes full. Uniformly sampled minibatches from the buffer are used to update the actor and critic network at each timestep during training. This benefits the learning as uncorrelated transition are used instead of successive transitions [28].

Since the policy is deterministic an agent may end up with a local solution or arrive at no solution at all. It is therefore necessary for the agent to explore the environment with new state-action pairs and evaluate whether they improve the current best policy π^*. The agent explores by letting the exploration policy μ' be influenced by a noise sampled from a noise process $\mathcal{N}$

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}. \tag{2.46}$$

$\mathcal{N}$ is selected as a Ornstein-Uhlenbeck process[61] in the original DDPG paper, but other processes may also be used. A Gaussian process is chosen as noise in this thesis with zero mean and $\sigma = 0.2$ where the probability density function is $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. There are works proposing alternative methods to ensure action space exploration, for instance by perturbing the network parameters instead, see [46]. This approach was shown to perform well and in some problems improved performance compared to action noise.

The complete DDPG algorithm is finally shown in Algorithm 2.

Algorithm 2 Deep Deterministic Policy Gradient (DDPG) — *Source: [28]*

```
Require: Actor network $\mu(s|\theta^{\mu})$ with weights $\theta^{\mu}$
Require: Critic network $Q(s, a|\theta^{Q})$ with weights $\theta^{Q}$
 1: Randomly initialise actor and critic network weights.
 2: Initialise target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
 3: Initialise replay buffer $R$
 4: for $episode \in \{1, 2, ..., M\}$ do
 5:     Initialise a random process $\mathcal{N}$ for action exploration
 6:     Receive initial observation state $s_1$
 7:     for $t \in \{1, 2, ..., T\}$ do
 8:         Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
 9:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
10:         Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
11:         Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
12:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13:         Update critic by minimising the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$
14:         Update the actor policy using the sampled policy gradient:
                $$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$
15:         Update the target networks:
                $$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
                $$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$
16:     end for
17: end for=0
```

Chapter 3

Explainable Artificial Intelligence

3.1 Terminology

The field of XAI involves different terms which currently are not standardised. [6] surveys and analyses current research literature and aims to move the field towards a unified terminology. Since XAI is rapidly changing a variety of terms are used interchangeably, for example transparency, intelligibility, interpretability, and explainability. The term XAI was coined in 2004 by Van Lent et al. in [62] describing how AI-controlled entities' behaviour in games can be explained [1]:

> Explainable AI can present the user with an easily understood chain of reasoning from the user's order, through the AI's knowledge and inference, to the resulting behavior.

Even though the field of XAI has been gaining momentum recently, the problem of explainability is not a new one. The term explainability has been used since mid-1970s when research was put into explainable expert systems [40].

The key findings from [6] are summarised as

- **Transparency** or transparent systems are, by their nature, interpretable without needing to provide explanations. Interpreting a system or providing explanations on the other hand are fundamentally different from a transparent system and should therefore not be interchanged with transparency.
- **Intelligibility** could be achieved through explanations and interpretations, where the type of user, their background, goal and current mental model are taken into consideration
- **Interpretability** is defined as a concept close to explainability. As examples, Shapley values, discussed in Section 3.4, and LIME, Section 3.3, are techniques used to

give insights into a model through interpretations.

- **Explainability**, or providing explanations is about improving the user's mental model of how a system works. This could be done through intuitive mediums such as natural language or visual clues.

Figure 3.1 shows an illustration of the relation between the terms.

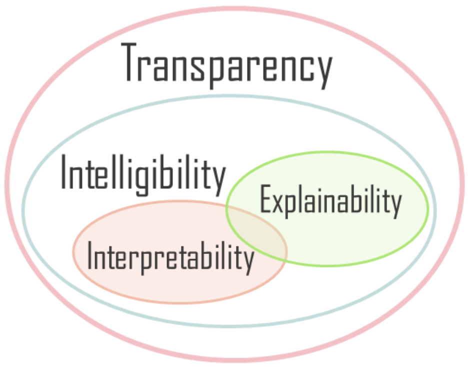


Figure 3.1: A venn diagram of the relationship between different terminology commonly used in XAI according to the findings from [6]. Note that interpretability intersects partly with explainability since some models may be interpretable without needing explanations. Figure from [6].

The findings are used throughout this thesis to provide an accurate and precise nomenclature.

3.2 Requisites for interpretable explanations

Establishing a foundation for what is required from an explanation is essential before diving into some suggested XAI methods. A simple yes or no answer may be sufficient to reason for a prediction in some simpler applications. However, in application requiring more sophisticated approaches, such as natural language processing, computer vision and robotics, binary explanations is of limited value. Instead, a visualization of where important features are present, like in a heatmap, could be more relevant. Since an explicit metric of explainability currently is unknown in the AI literature, most XAI methods are based on the researchers own intuition of what a good explanation constitutes. This could limit the advancement of these methods as understandings from explanations varies greatly with the depth of knowledge in these systems. For this reason, Miller [38] argues that multiple viable strategies to attain interpretable explanations in AI should be built with consideration from known research in philosophy, psychology, and cognitive sciences. From these fields it is known that people exert biases, social expectations and explanation selection whenever reasoning is presented. Miller pinpoints four major findings in his review which he claims that researchers and practitioners currently are unaware of:

1. Explanations should explain why event A happened instead of B. People seldom ask why event A occurred, but rather seek reasons for it over another case.

2. Selective explanations induce biases. Humans select one or two reasons for an event from a pool of infinite possible causes. It is therefore, subconsciously, rarely expected a complete explanation for an event. This has the unavoidable effect of inducing cognitive biases whenever a reason is selected and being presented.
3. Statistical relationships are ineffective at delivering an explanation to humans. While they do have some value, causal relations provide more meaningful reasoning compared to probabilities. The net takeaway from this point is the accompanied value of using both at the same time and avoid likelihoods alone.
4. An explanation is a conversational transfer of knowledge presented in belief of the recipient's view. This implies that an explanation is a social interaction conveyed through a communicative medium, that being e.g. an image, natural language, body language, text etc. or any combination of these.

Applying practices inferred from these findings will likely increase the explanation value for experts and the layman. This is essential for societal trust to future AI driven systems as it is taking over evermore tasks in the industry, infrastructure, transportation, medicine and ultimately peoples daily lives. If - or for that matter when - AI systems are handed these responsibilities, it is vital that their decisions are deeply understood such that the public opinion maintains trust even in the off chance of failure. The XAI methods presented next aims to open the black box models and lay the foundation for future applications.

3.3 LIME

Local Interpretable Model-agnostic Explanations (LIME) is an explanation framework created to guide experts on the behaviour of a trained black box model[59]. It aims to answer the question of "why should I trust the model?" by finding an interpretable model that is locally reliable to the classifier. It also supports regression models. The framework is open sourced and available at the lead author's GitHub [58]. Summarized, LIME approximates the black box model around an input by assuming that the input space is locally accurate to the classifier. This means that the explanation are made at the individual level.

A vector $x \in \mathbb{R}^d$ represents the unaltered input while $x' \in \{0, 1\}^{d'}$ symbolizes a binary vector for the explainable representation. This could for instance be whether a set of pixels are present or not. An explanation is defined as a model $g \in G$, where G denotes all possible interpretable models. The model g could for instance be a linear model, decision tree or a falling rule list with "IF-THEN" statements. This implies that it needs to facilitate simple, interpretable visual or textual explanations. Furthermore, as not all possible explanation models g are simple enough for human interpretability, the authors introduce $\Omega(g)$ to be a measure of complexity of the explanation model. As an example, for a decision rule, $\Omega(g)$ could be the number of statements needed in a "IF-THEN" rule or the number of non-zero weights for a linear model. The fewer statements or non-zero weights needed, the simpler the model and less penalty is introduced by the $\Omega(g)$ term. The classifier being explained, $f : \mathbb{R}^d \rightarrow \mathbb{R}$, outputs the probability $f(x)$ that an input x belongs to a specific class[1]. z is a sample of x and $\pi_x(z)$ is used as a measure of proximity between z to x. The

[1]This is slightly different from conventional notation where f is defined with k-outputs, $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$. The

locality in LIME is obtained through the weighting term π_x. Finally, $\mathcal{L}(f, g, \pi_x)$ is defined as a measure of how inaccurate g approximates f. This could for instance be a distance measure between f and g. LIME tries to minimize the objective function stated as

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \, \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.1}$$

in order to obtain a valid local approximation of f while reducing the complexity of g to ensure human interpretability. LIME is model-agnostic, meaning that the explanation is separated from the choice of model. The objective function in (3.1) establishes this foundation by conveniently allowing individual choices for G, $\mathcal{L}$ and Ω. The term $\mathcal{L}(f, g, \pi_x)$ is approximated, since it should be independent of f, by drawing non-zero elements of x' uniformly at random. A such perturbed sample $z' \in \{0, 1\}^{d'}$ is passed through the model to obtain $f(z)$, which is the prediction probability of z' belonging to the specified class. Given enough perturbed samples, (3.1) is optimized to get an explanation of x, namely $\xi(x)$.

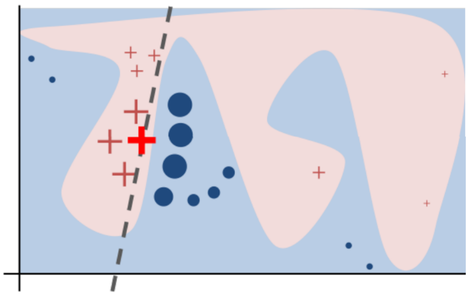


Figure 3.2: The intuition behind LIME. The nonlinear decision function is represented in pink. LIME learns a linear model and explains a prediction (red cross) through the explanation model represented by the dashed line. Notice that it is only locally faithful, not globally. Figure from [59].

Linear LIME

Linear LIME utilises a linear explanation model $g \in G$, such that $g(z') = w_g \cdot z'$. The square loss, also known as quadratic loss, is used for $\mathcal{L}$ and $\pi_x = \exp\left\{(-\frac{D(x,z)^2}{\sigma^2})\right\}$ is the measure weighting the proximity between z and x where D is a distance function. The D is defined in the code as the l^2 norm, with $\sigma = 0.75 \cdot \sqrt{\text{number of columns}}$. The term is finally set to

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(x)(f(z) - g(z'))^2. \tag{3.2}$$

Algorithm 3 finds the weights w_g used in the linear model g with a least squares regression (Lasso).

author rather defines $f(x)$ to be the prediction probability of chosen class instead, simplifying the notation in the process.

Algorithm 3 Sparse Linear Explanations using LIME — *Source: [59]*

```
Require: Classifier $f$, Number of samples N
Require: Instance $x$ and its interpretable version $x'$
Require: Similarity kernel $\pi_x$ and the length of explanation $K$
1: $Z \leftarrow \{\}$
2: for $i \in \{1, 2, ..., N\}$ do
3:     $z'_i \leftarrow \text{sample\_around}(x')$
4:     $Z \leftarrow Z \cup < z'_i, f(z_i), \pi_x(z_i) >$
5: end for
6: $w \leftarrow K - Lasso(Z, K)$          ▷ using $z'_i$ as features, $f(z)$ as target
7: return $w$
```

3.4 SHapley Additive exPlanations (SHAP)

SHapley Additive exPlanations or SHAP for short, is a proposed framework to interpret predictions provided by a model [33]. It is a game theoretic approach to explain any black box model, and is therefore model agnostic. The authors have open sourced their work and published it freely available at GitHub [32]. The authors propose SHAP values, similar to Shapley values, as a unified measure of feature importance. The method unifies the ideas from LIME and the game theoretic Shapley values. As such, a brief introduction to Shapley values follows.

3.4.1 Shapley values

A brief introduction to Shapley values is given as it forms the basis on which the SHAP method is built upon. The Shapley values were introduced by Lloyd S. Shapley in 1953 [53] as a concept in cooperative game theory. A Shapley value is assigned to each player in a game stating how important they are in the cooperation to a surplus. It provides one way of fairly distributing the payout to players based on their contribution in the game. The coalition game is described using the set N of n players. Let a coalition of players S be a subset of N, i.e $S \subseteq N$, and define a payout function v, describing the expected payout to each possible coalition. In a coalition game (v, N), the amount player i collects is the Shapley value

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S)]. \tag{3.3}$$

The term $[v(S \cup \{i\}) - v(S)]$ can be interpreted as calculating the payout to a coalition with player i minus the payout without player i. The term is weighted by $\frac{|S|!(|N|-|S|-1)!}{|N|!}$ and together with the sum $\sum_{S \subseteq N \setminus \{i\}}$, represents, for each player, the average contribution over different possible permutations in which the particular coalition can be formed. Summarized, the Shapley value for player i (3.3) states the importance of the player by comparing the payout with and without them in a coalition of players. Since the order in which the players contributes may affect the payout, the contribution is calculated with all possible permutations of the coalition, across all possible coalitions.

The Shapley value obeys properties important to obtain a fair and unique distribution. It is also the only attribution method that satisfies these desirable properties [19]. The original paper [53] states these properties in the definition as *symmetry*, *efficiency* and *law of aggregation* (linearity). Finally, the property of a dummy or null player is explicitly stated in the paper's definition section. These properties are

Symmetry

If two players are equal and contribute the same, then they receive equal payout. Formally, if $v(S \cup \{i\}) = v(S \cup \{j\})$ then $\phi_i(v) = \phi_j(v)$.

Efficiency

The sum of the Shapley values of all players equal the payout for the total coalition

$$\sum_{i \in N} \phi_i(v) = v(N) \tag{3.4}$$

Law of aggregation (linearity)

If two independent games are combined, then the payout equal the payout sum for each individual game

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w) \tag{3.5}$$

Null player

A player that does not contribute to the payout receives no payout. If $v(S \cup \{i\}) = v(S)$ for all coalitions S without i, then $\phi_i(v) = 0$

3.4.2 SHAP Method overview

Even though the concept of Shapley values was proposed in the field of cooperative game theory, it may be used as a means to explain a prediction. By assuming that each feature is a player and the prediction is the total payout, then the Shapley value tells how much each individual feature contributed towards the prediction. Unfortunately, calculating the Shapley value going through all possible combinations of features is computationally heavy. Computing the marginal contribution of every feature to every coalition is $O(2^{|N|})$ [19] which quickly becomes infeasible. SHAP values are based on the Shapley values and obeys their properties, adding strong mathematical theory behind it. An illustration of the SHAP values is shown in Figure 3.3 The authors propose a model agnostic method to obtain the SHAP values, namely Kernel SHAP.

Kernel SHAP

Kernel SHAP is presented in the SHAP paper[33] as model agnostic approximation method to obtain Shapley values. Kernel SHAP uses (3.1) to obtain the Shapley values, but unlike

LIME, avoids heuristically choosing the parameters for loss function $\mathcal{L}$, weighting kernel π_x and regularization term Ω. Rather, they are chosen such that the properties of Shapley values are retained. These are shown to be

$$\Omega(g) = 0, \tag{3.6}$$

$$\pi'_x(z') = \frac{(M-1)}{(M \text{ choose } |z'|)|z'|(M-|z'|)}, \tag{3.7}$$

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in Z} (f(h_x(z')) - g(z'))^2 \pi'_x(z'). \tag{3.8}$$

M is the maximum number of features in a coalition, being the size of set N, $|N|$, in the original description of Shapley values. M choose $|z'|$ is the binomial coefficient $\binom{M}{|z'|} = \frac{M!}{|z'|!(M-|z'|)!}$. The explanation model g in (3.8) takes a linear form. The kernel π'_x in Kernel SHAP essentially merges the theory of Shapley values with the model agnostic approach of LIME. Even though Kernel SHAP is model agnostic like LIME, it is unfortunately computationally heavy and therefore slow.

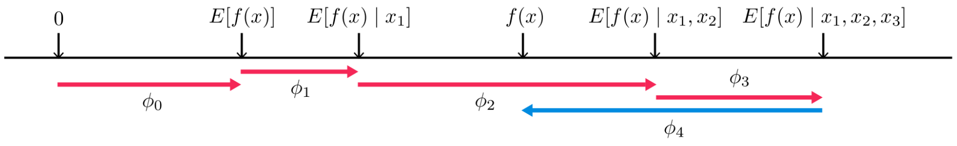


Figure 3.3: The SHAP values decomposes the output from a function f into a sum of the contribution ϕ_i of the features. Because of the property of *efficiency*, the contributions sums up to the output, i.e. $\sum_i^N \phi_i = f(x)$. Figure from [31].

Chapter 4

Path following for kinematic vehicle model

4.1 Lateral vehicle kinematics

Assumptions

A four-wheeled vehicle can, under certain assumptions, be approximated by a bicycle model. The lateral vehicle model described in [49] is a simplified vehicle where the two front wheels are represented as a single point, A. The rear wheels are similarly represented as one wheel at point B, nearing the vehicle to a bicycle. The model is constrained to planar motion and has a rigid body. Low velocity is assumed such that slippage of the tires can be neglected. This means that the velocity vector at the front wheel makes an angle equal to the steering angle, δ_f. The rear wheel angle δ_r is adjustable in the dynamics presented in [49], however it is set to zero in this thesis to reduce the number of control actuators to one. Finally, the front wheel saturates at angles $\delta_{f_{min}} = -1$ rad $\approx -57.3°$ to $\delta_{f_{max}} = 1$ rad $\approx 57.3°$. Summarised, the model is a mathematical description of the motion based on geometric relationships of the system without considering forces.

Equations of motion

The model considered is depicted in Figure 4.1 where the distance between the front wheel at A and the rear wheel at B is the distance of the wheelbase $l_f + l_r$. The center of gravity is at the point indicated by CG. The planar motion is described in the earth-fixed North-East-Down (NED) coordinate system where the x axis points towards true North, the y axis towards East and the z axis down into the plane. The positive rotation is clockwise as a consequence when following the right-hand screw convention. The (x, y) position coordinates are located at CG with the heading (yaw) angle ψ describing the orientation of the vehicle. The velocity at CG U is composed of the body surge u and sway v velocities and produces a sideslip (drift) angle β with the longitudinal axis. Finally, the course angle is defined by $\chi = \psi + \beta$.

Referring to Figure 4.1, the kinematic equations of motion of the vehicle can be expressed as

$$\dot{x} = U\cos(\psi + \beta) \tag{4.1}$$

$$\dot{y} = U\sin(\psi + \beta) \tag{4.2}$$

$$\dot{\psi} = \frac{U\cos\beta}{l_f + l_r}(\tan\delta_f - \tan\delta_r) \tag{4.3}$$

$$\beta = \arctan\left(\frac{l_f\tan\delta_r + l_r\tan\delta_f}{l_f + l_r}\right) \tag{4.4}$$

where δ_r is set to zero in the figure.

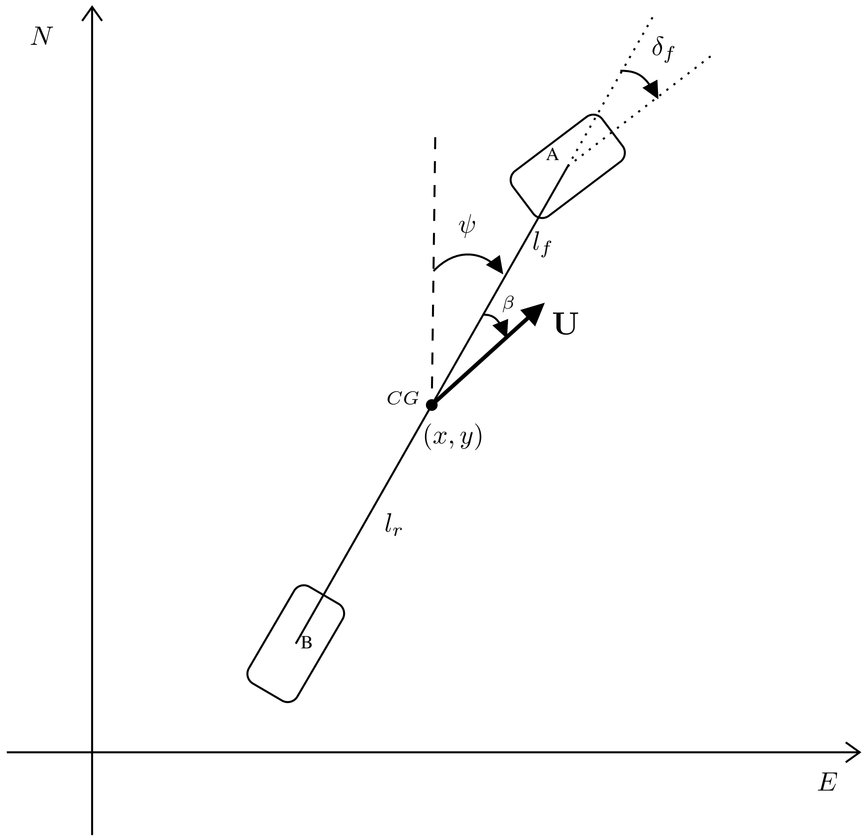


Figure 4.1: The lateral vehicle model expressed in North-East-Down (NED) reference frame. The front wheel is attached at point A and can be turned by a controller making an angle δ_f in relation to the (yaw) heading angle ψ. The velocity vector $\mathbf{U}$ is not always aligned with the heading as the vehicle experiences sideslip with angle β when the sway velocity $v \neq 0$. Figure inspired by [8].

4.2 Cross-track error for straight-line path following

As formulated in [12], a straight-line path defined between two waypoints $\mathbf{p}_k^n = [x_k, y_k]^\top \in \mathbb{R}^2$ and $\mathbf{p}_{k+1}^n = [x_{k+1}, y_{k+1}]^\top \in \mathbb{R}^2$ is considered. The straight-path following problem is depicted in Figure 4.2 where the vehicle is moving at speed $U = |u| + |v| = \sqrt{\dot{x}^2 + \dot{y}^2}$. A path-fixed reference frame is defined with origin in $\mathbf{p}_k^n$ with a positive rotation α_k relative to NED. The coordinates of the vehicle at time t is $\mathbf{p}^n(t) = (x, y)^\top$ and can be expressed in the path-fixed reference frame as the error vector

$$\boldsymbol{\epsilon} = \mathbf{R}_p(\alpha_k)^\top (\mathbf{p}^n(t) - \mathbf{p}_k^n) \tag{4.5}$$

where

$$\mathbf{R}_p(\alpha_k) = \begin{bmatrix} \cos(\alpha_k) & -\sin(\alpha_k) \\ \sin(\alpha_k) & \cos(\alpha_k) \end{bmatrix} \in SO(2). \tag{4.6}$$

The error vector $\boldsymbol{\epsilon} = [x_e, y_e]^\top$ expresses the along-track distance x_e and the cross-track error y_e. Since the problem is to follow a straight-path, only y_e is of interest. From (4.5) y_e is found as

$$y_e(t) = -[x(t) - x_k]\sin(\alpha_k) + [y(t) - y_k]\cos(\alpha_k) \tag{4.7}$$

where the objective is to minimize this error such that

$$\lim_{t \to \infty} y_e(t) = 0 \tag{4.8}$$

meaning the vehicle has converged to the straight-path.

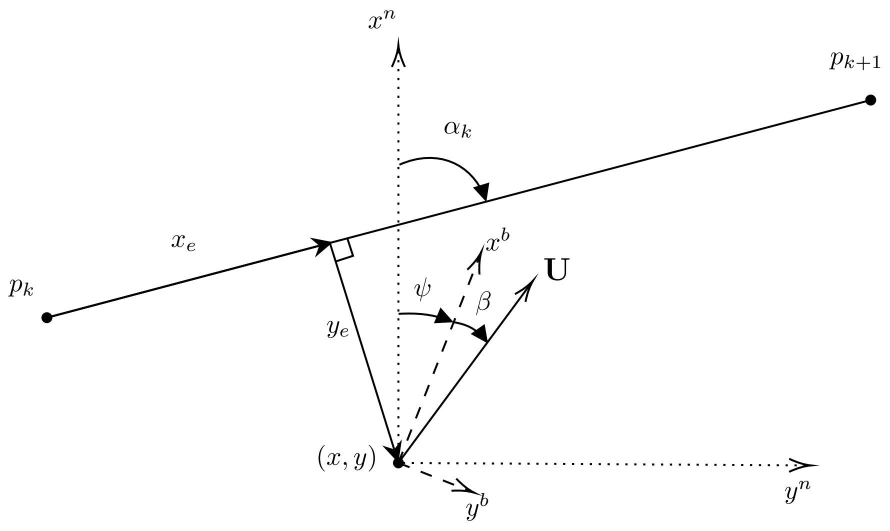


Figure 4.2: Illustration of the path following problem. The main geometric relationships are included and expressed in the NED coordinate system. The heading angle ψ defines the angle between NED and the Body coordinates. Figure inspired by [25].

4.3 Curved path following

This section is inspired by [12, 35]. For curved path following, a predefined parametrised path is assumed available. A possible representation of the parametrised path for a vehicle operating in 2D space is

$$\mathbf{p_d}(\omega) = [x_d(\omega), y_d(\omega)] \tag{4.9}$$

where ω is the parametrisation variable. The first and second order derivatives of $\mathbf{p}_d$ with respect to ω are denoted $\mathbf{p}_d'$ and $\mathbf{p_d}''$ [12].

The angle of the path changes over the curve unless it is a straigh-line. The path angle is therefore defined as the tangential angle with respect to the derivatives of the parametrised path

$$\alpha_p(\omega) = \text{atan2}(y_d'(\omega), x_d'(\omega)). \tag{4.10}$$

This is also visualised in Figure 4.3

As with the straight-line path following, the geometric task is to force vehicle to converge to the desired path, i.e.

$$\lim_{t\to\infty} y_e(t) = 0. \tag{4.11}$$

The cross-track error can be found by inserting the path angle $\alpha_p(\omega)$ and $\mathbf{p_d}(\omega)$ in (4.7) as follows

$$y_e(t) = -[x(t) - x_d(\omega)]\sin(\alpha_p) + [y(t) - y_d(\omega)]\cos(\alpha_p) \tag{4.12}$$

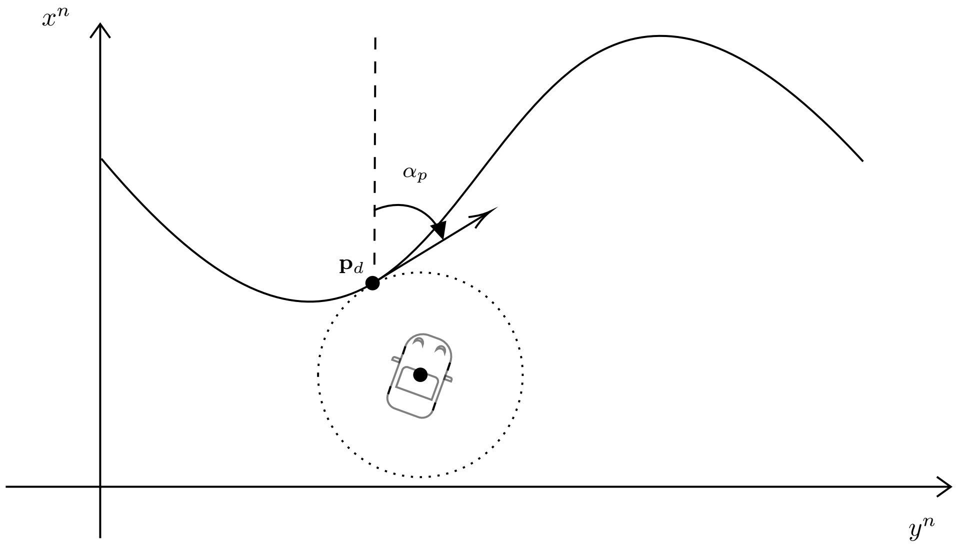


Figure 4.3: Illustration of the path tangential angle at the closest distance to the vehicle in a curved path following problem. Figure inspired by [35].

The closest point on the curve to the vehicle can be located by finding the path variable ω expressed as the convex optimisation problem

$$\min_{\omega} f(\omega) = (x - x_d(\omega))^2 + (y - y_d(\omega))^2 \tag{4.13}$$

given the vehicles current position (x, y) [35]. This can for example be solved at each time instant using the Newton's method to find the ω that satisfies $\frac{\mathrm{d}f(\omega)}{\mathrm{d}\omega} = 0$. The previous ω can used to speed up the computation when implementing the method.

4.4 Training an agent using DDPG

The straight-path following problem is the task of following a predefined path independent of time. There are a few methods for achieving this objective where a commonly used approach is by using line-of-sight (LOS) guidance[12]. The LOS guidance scheme requires a cascading control system hierarchy where lower level controllers are responsible for motion control. The LOS controller receives estimates of the vehicles states and gives set-points to the lower level controllers, resulting in the cascading structure. Due to the high nonlinear nature of the problem and uncertainty in the modelling of the vehicle other approaches have been applied to solve the straight-path following problem. One such is a DRL-based framework proposed in [36], which the following section is utilising. The relationship between the DDPG agent and the environment is shown in Figure 4.4.

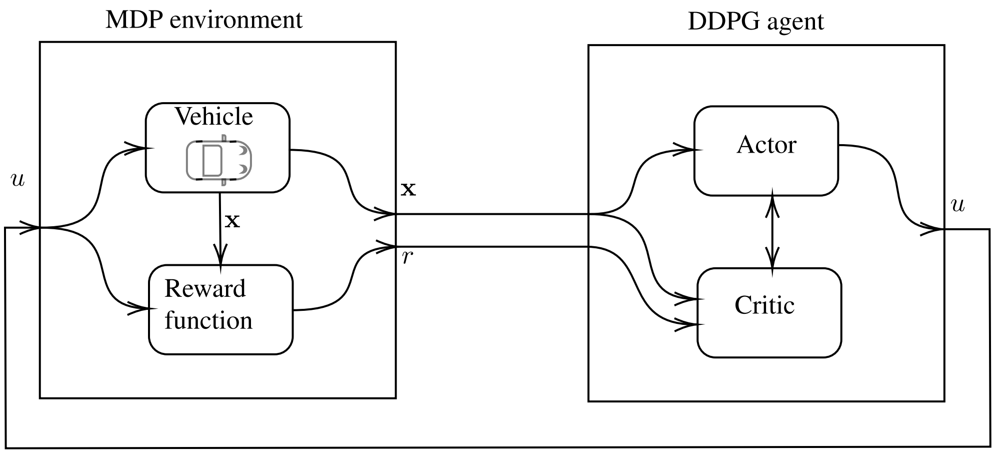


Figure 4.4: A schematic of the interactions between the DDPG and the environment modelled as MDP. The agent receives the state $\mathbf{x}$ and the reward r and computes the action u sent to the vehicle. Notice that the vehicle model is hidden for the agent.

DDPG setup

The implementation of DDPG used in the thesis relies on Udacity's version[60]. The pseudocode is given in Algorithm 2 and the theory introduced in Section 2.7.1. In the implementation, the Ornstein-Uhlenbeck process is changed to a Gaussian noise process for generating noise to the actions during training. The noise is added to ensure exploration is performed when the agent is in the learning phase. Next, the parameters used for the actor and critic networks are similar to the implementation in the paper proposing DDPG[28] with some modifications. The actor network is responsible for the policy $\pi(\mathbf{x})$ which outputs the deterministic action u. The actor is built by connecting the input states using two hidden layers with both 50 nodes connected to a single action node at the output layer. The Relu activation function is used between the hidden layers with a hyperbolic tangent function at the output ensuring a value between -1 and 1.

$$\begin{aligned}\mathbf{f}_1 &= \text{Relu}\left(\mathbf{W}_0\mathbf{x} + \mathbf{b}_0\right)\\ \mathbf{f}_2 &= \text{Relu}\left(\mathbf{W}_1\mathbf{f}_1 + \mathbf{b}_1\right)\\ u = \pi(\mathbf{x}) &= \tanh\left(\mathbf{W}_2\mathbf{f}_2 + \mathbf{b}_2\right)\end{aligned}$$

The critic network takes the state $\mathbf{x}$ and actor action output u as input. The two hidden layers consist of 300 and 400 nodes. The output layer consist of a single node, giving out the Q-value. This results in the following architecture:

$$\begin{aligned}\mathbf{f}_1 &= \text{Relu}\left(\mathbf{W}_0\mathbf{x} + \mathbf{b}_0\right)\\ \mathbf{f}_2 &= \text{Relu}\left(\mathbf{W}_{1a}\mathbf{f}_1 + \mathbf{W}_{1b}u + \mathbf{b}_1\right)\\ Q(\mathbf{x}, u) &= \mathbf{W}_2\mathbf{f}_2 + \mathbf{b}_2\end{aligned}$$

The weights W_i and biases b_i are updated according to Algorithm 2 based on the received feedback from the reward. The replay buffer's maximum size is set to 10^6 transition samples where the agent randomly samples minibatches of size 128. The discount factor γ is set to 0.99 and the soft target update constant τ is set to 10^3. The actor or policy network uses a learning rate of 10^{-4} while the critic or value function network uses a learning rate of 10^{-3}. Finally, the critic optimiser uses L2 weight decay with value 10^{-2} to penalise larger weights in the network.

Designing a state representation

The kinematic equations from Section 4.1 describes the motion of the vehicle and are hidden for an external observer. Since the path angle α_k and path-fixed coordinate system $\mathbf{p}_k^n$ as defined in Section 4.2 are known and assuming that the heading ψ and position $\mathbf{p}^n(t) = (x, y)^\top$ is measured at each time instant t, a subset of the system's states can be used to conveniently represent the problem objective.

Combined with the cross-track error y_e another state is exploited, namely the error signal in the heading angle relative to the path angle, $\tilde{\psi} = \psi - \alpha_k$. The relationship is shown in Figure 4.5. This measure indicates the deviation of the heading in relation to the direction of the path. If the cross-track error is zero and $\tilde{\psi}$ is positive, then the vehicle should intuitively turn counterclockwise ($\delta_f < 0$) in order to align with the path. Likewise, if $y_e = 0$ and $\tilde{\psi} < 0$ then a turn clockwise should be performed. Heavy disturbance and a large sideslip β will influence the course χ and may render a different strategy necessary to maintain a trajectory on the straight path. Disturbances are however not included as a part of the experiment.

The state vector available for the learning algorithm is $\mathbf{x} = [y_e, \tilde{\psi}]^\top$. The learning algorithm utilised is the DDPG introduced in Section 2.7.1 which trains an agent to maximize expected reward. The agent observes the state $\mathbf{x}$ and actuates the front wheel by the control action $u_t \to \delta_f$ which saturates by the constraint of $\pm 1\,\text{rad} \approx \pm 57.3°$.

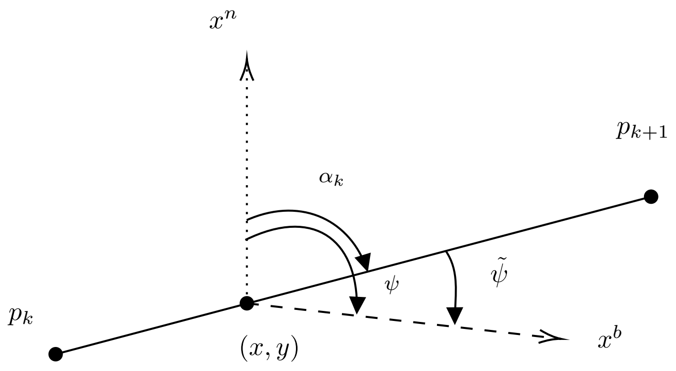


Figure 4.5: The geometric relationship between $\tilde{\psi}$, ψ and α_k. Positive rotation in the direction as indicated.

Reward function

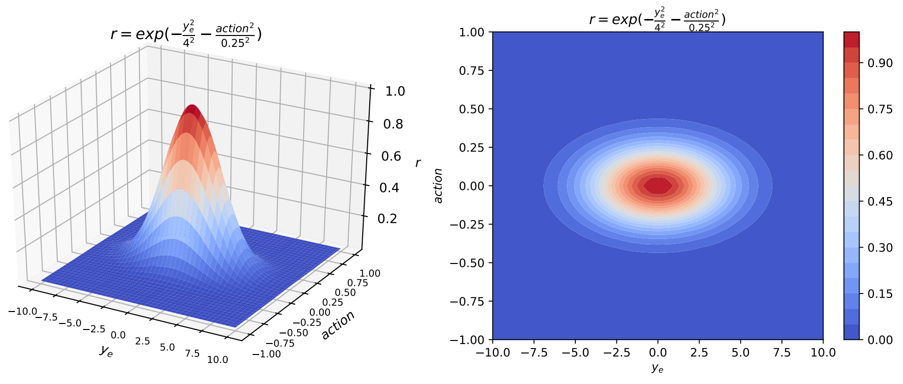
The vehicle model is treated as a black box where the learning agent only observes the state input $\mathbf{x}_t$, the control action u_t and the consequent reward r_t. The agent updates its internal weights based on the reward received according to the DDPG algorithm. It is therefore crucial that the feedback acquired actively reflects the deviation between the current state-action pair and the desired goal. By rewarding the agent little when it's far from the desired state and more when it's close, appropriate behaviour is encouraged. This technique is inspired by animal training and is called reward shaping [65]. As highlighted in [37], the reward function should be converted into error signals by using implicit domain knowledge. This has been the primary motivation to use error-signal state vector $\mathbf{x} = [y_e, \tilde{\psi}]^\top$ available for the learning algorithm. The domain knowledge of driving the cross-track error y_e towards zero is implemented in the reward function by rewarding the agent the most at $y_e = 0$ and decrease the reward as the error increases. The Gaussian function e^{-x^2} serves this purpose quite well as it is everywhere differentiable (smooth), has it's peak $\lim_{x\to 0} e^{-x^2} = 1$ at $x = 0$ and goes towards zero as $x \to \pm\infty$. Next, the agent should perform action of lower magnitudes to avoid sharp turns. This is added in the reward function by including the term $e^{-action^2}$. The agent should obey these two goals simultaneously and this is enforced by multiplying the two reward function resulting in the following reward function

$$r(\mathbf{x}, action) = exp(-\frac{y_e^2}{4^2} - \frac{action^2}{0.25^2}) \tag{4.14}$$

which is plotted in Figure 4.6. Note that action and the control input u_t refers to the same variable. The coefficients in (4.14) are found by trial and error when performing training of the agent.

Training

The agent is trained with episodes where the vehicle is initialised with uniform probability within $\pm 10\,\mathrm{m}$ of the straight path, i.e. $-10 \leq y_e \leq 10$. The heading is initialised such that



(a) Surface plot of the reward function

(b) Contour plot of the reward function

Figure 4.6: Plots of the reward function $r = \exp\Big\{(-\frac{y_e^2}{4^2} - \frac{action^2}{0.25^2})\Big\}$. The agent receives greater reward when the cross-track error y_e and the control $action$ is simultaneously close to zero. The peak reward of $r = 1$ is found at $(0, 0)$. Note that the reward is always positive, thereby encouraging the agent to be within bounds and discourage interruption.

the heading error $\tilde{\psi}$ is uniformly initialised $\pm\frac{\pi}{2}$ rad, giving the agent a sense of direction. The velocity U is a constant set to $3\,\mathrm{m/s}$, the mass $m = 5\,\mathrm{kg}$ and the distance between the front axle to the center CG is $l_f = 1\,\mathrm{m}$ and center to rear axle $l_r = 1\,\mathrm{m}$. The straight path is a line passing through the origin of NED at an angle making $\alpha_k = 45° = \frac{\pi}{4}$ rad.

The agent is trained by simulating epochs of maximum length 1000 timesteps with $\tau = 0.02\,\mathrm{s}$ amounting to 20 seconds of real time. The state $\mathbf{x}_t$, action (u_t), reward r_t and the next state x_{t+1} are saved at time t in the replay buffer R. Samples are randomly drawn from the buffer and used to update the weights (θ^{μ} and θ^{Q}) following the scheme of Algorithm 2 - Deep Deterministic Policy Gradient (DDPG). The epoch is terminated when $|\tilde{\psi}| > \frac{7\pi}{12}\,\mathrm{rad} = 105°$ or the cross-track error surpasses the threshold $y_e > 20\,\mathrm{m}$. The vehicle is allowed an angle slightly larger than perpendicular such that the agent may explore angles in the near vicinity of normal to the path. The resulting training progression is shown in Figure 4.7.

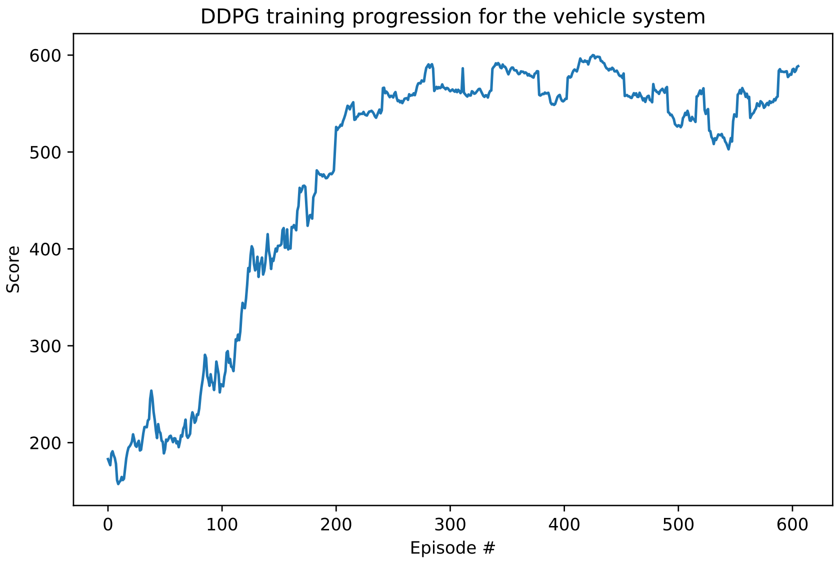
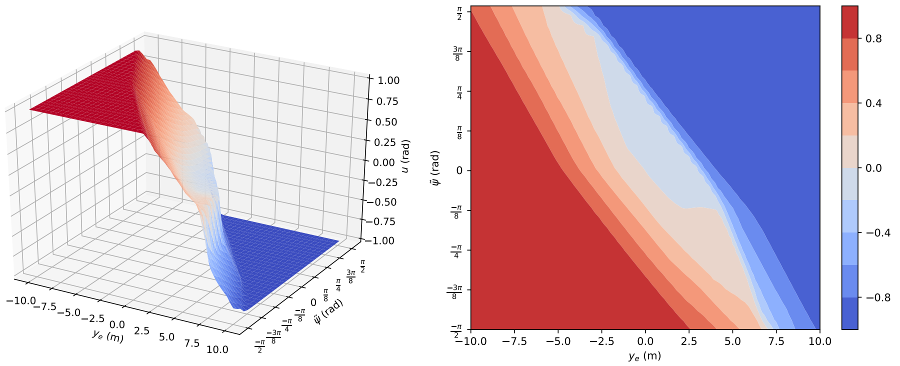


Figure 4.7: The total score over episode number for the DDPG agent during training in Section 4.4. The agent trained for a total of 650 episodes. Note that the plot is smoothed by a moving average with a window of 25.

Policy

DDPG finds the controller input from the policy $\pi(\mathbf{x_t}) = u_t$. Since the policy is defined $\pi : \mathbb{R}^2 \to \mathbb{R}$ it can be visualised as seen in Figure 4.8.



(a) Surface plot of the DDPG policy

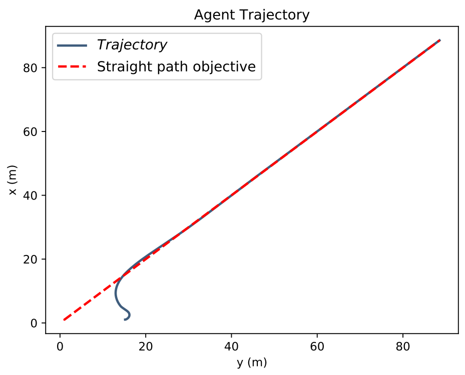
(b) Contour plot of the DDPG policy

Figure 4.8: Plots of the DDPG policy.

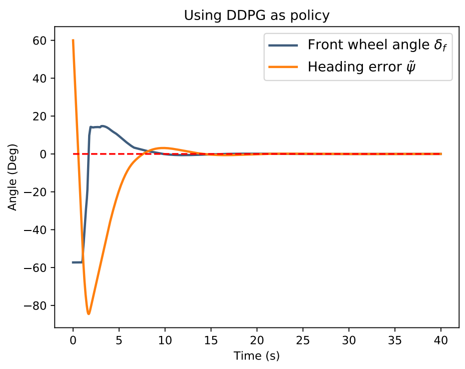
As observed in the figure, the policy appears to symmetrical about the oblique line passing through the desired state $\mathbf{x} = [0, 0]^\top$. This suggests that the agent has learned almost symmetrical behaviour about the straight-line path.

Performance of DDPG agent

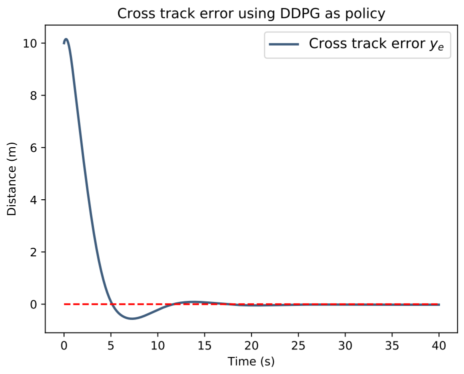
A set of initial conditions are determined in order to compare the performance between agents. These are close to the boundary of the trained region for the DDPG agent such that the simulation provides an indication of how the agent performs near the edge of the policy. y_e is set to $10\,\mathrm{m}$ and the heading is set so the heading error is $\tilde{\psi} = \frac{\pi}{4}\,\mathrm{rad} = 45°$. The agents are allowed 2000 steps in the environment to make steady state performance comparisons possible. The action noise used to explore the action space during training of the DDPG is now turned off since the agent is no longer training. Figure 4.9a shows the DDPG agent's trajectory starting with the initial conditions described in this section. The agent starts by turning such that the heading points almost orthogonal towards the straight path before it straightens up the heading error, $\tilde{\psi} \approx 0$, to continue along the objective line. The cross-track error, seen in Figure 4.9c, fluctuates slightly around zero meaning that a sub-optimal policy is found since it does not remain at zero. The error should vanish if the policy converges to the optimal policy, $\pi(\mathbf{x}) \to \pi^*(\mathbf{x})$. This could however require orders of magnitude more training epochs in order for the agent to reach such a policy and it is also not guaranteed. Another possible solution is to create a steeper Gaussian reward function, but this could impact the convergence rate, meaning it in principle needs more training. A more feasible solution is to supply the agent with integral action to the cross-track error y_e as proposed in [36]. The augmented state vector would then take the form $\mathbf{x} = [y_e + k_i \int y_e d\tau, \tilde{\psi}]^\top$.



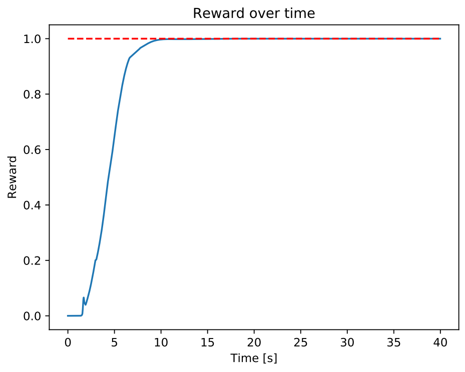
(a) Trajectory



(b) Vehicle states



(c) Cross-track error



(d) The reward gathered

Figure 4.9: Plots when initialised with $y_e = 10$ and $\tilde{\psi} = \frac{\pi}{4}$ rad

4.5 Model tree regression

The DDPG agent in Section 4.4 reaches its goal of zero cross-track error with some margin, but its policy is nonlinear and acts as a black box. This is because the function expression is dependant on the many internal weights of the neural network. This DDPG policy is visualiseable since $\pi : \mathbb{R}^2 \rightarrow \mathbb{R}$, but this is not the case in general since the input vector and the output could have any number of states in principle. This means that the policy could be highly dimensional and perceived as a black box for humans. To make the DDPG policy transparent, the idea is to transform the nonlinear policy into a set of regions where linear relationships are found using a model tree. This means that the nonlinear policy have been transformed into a *piecewise-linear policy*. The resulting tree indicates how the policy behaves in different regions and is traceable from input to output, in essence opening up the black box. An introduction to model trees is provided in Section 2.4. The model tree used in this thesis is built upon the implementation by Anson Wong [66]. The code is adapted to support polynomial regression at the leaf nodes with rendering of the function expressions when exporting the model tree as a figure. Support for rendering the linear function expressions at the leaf nodes is also added. Finally, the True/False terms are inserted in the model tree to avoid confusion of how it should be read. The implementations uses the Δ-criterion in the WORTHSPLITTING function in Algorithm 1. The cost function is set to mean squared error (MSE), which is used in both WORTHSPLITTING and SPLIT. Further, two more constraints are available in WORTHSPLITTING: a maximum depth and a lower bound of samples that must be present in each leaf. This ensures both that the tree does not grow unbounded in depth and that each split region is larger than the given minimum value.

Policy sampling

In order to capture the complete policy from the DDPG policy, a uniformly distributed sampling of the DDPG policy is performed. The policy is sampled across all possible combinations of the input states $\mathbf{x}$ from $y_e = -10$ to $y_e = 10$ with step 0.2 and from $\tilde{\psi} = -\frac{\pi}{2}$ to $\tilde{\psi} = \frac{\pi}{2}$ with step 0.03. This results in a total of $100 \cdot 104 = 10400$ uniformed samples of the policy. A visualisation is shown in Figure 4.10. The input and consequent policy action $u = \pi(\mathbf{x})$ is saved as training data for the model tree[1]. A final constraint of a lower bound of minimum 100 samples per leaf node is added. This is to ensure that the trees generalise to regions of a minimum size.

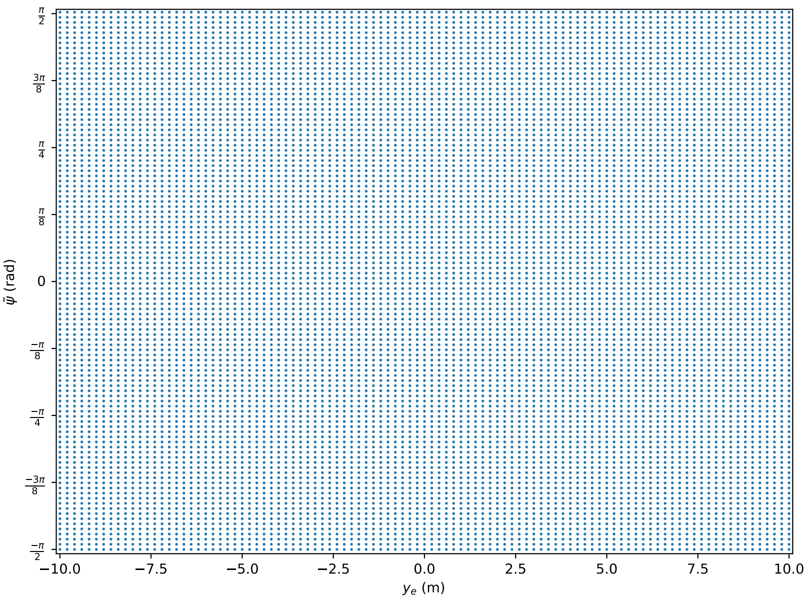


Figure 4.10: The uniformed sampling points used to collect a training set for the model tree.

[1]There are multiple ways to gather training data for the model tree. The first tried approach consisted of running the DDPG agent with multiple random initial conditions and save the input-output pairs. This worked on the surface level, but did not capture the outliers of the DDPG policy and was as a result discarded as a strategy for training the model tree.

4.5.1 Linear Model Tree depth 2

In this part, a model tree with a linear function at the leaf node is considered. This is referred to as Linear Model Tree (LMT)[2]. The tree is trained following the procedure introduced in Section 2.4 where the input-output pairs of the DDPG policy are uniformly sampled. The maximum depth is set to 2 giving a total of 4 leaf nodes. The resulting model tree is displayed in Figure 4.11. Like in i Section 4.4, the LMT depth 2 policy may be visualised. As seen in Figure 4.12, the LMT output resembles the DDPG agent's output seen in Figure 4.8, though with some important differences. For instance, the output is not strictly bounded between $[-1, 1]$ which is the saturating limit of the vehicle steering angle. This means that the model tree may need to be deeper to allow a better approximation of the DDPG agent. The second difference is the noticeable jumps between the linear regions. This shows that the LMT depth policy is *discontinuous* piecewise linear which in some instances could be undesirable.

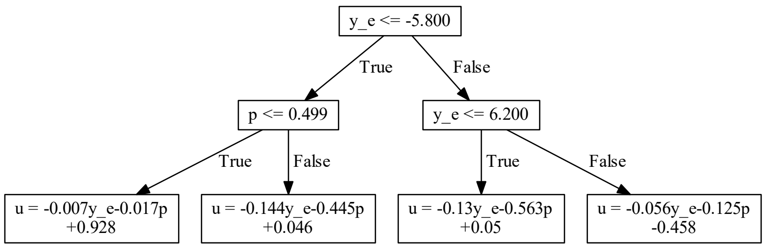
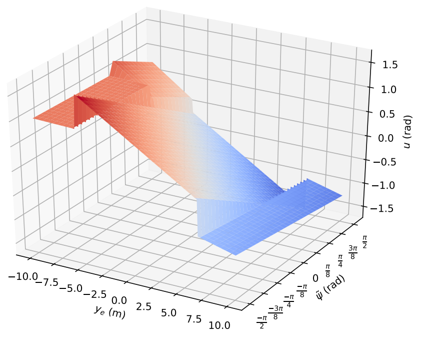
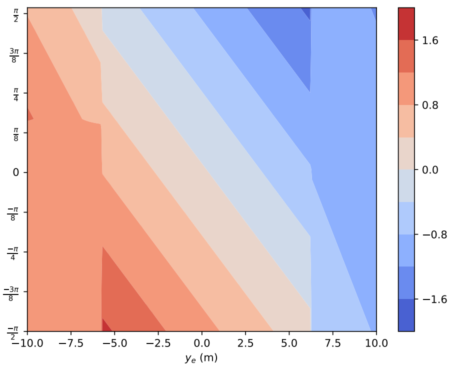

y_e <= -5.800
True
False
p <= 0.499
y_e <= 6.200
True
False
True
False
u = -0.007y_e-0.017p +0.928
u = -0.144y_e-0.445p +0.046
u = -0.13y_e-0.563p +0.05
u = -0.056y_e-0.125p -0.458

Figure 4.11: The model tree with depth 2. y_e is state y_e and p is $\tilde{\psi}$. u is the action from the tree used to control the vehicle.



(a) Surface plot of the Model Tree policy



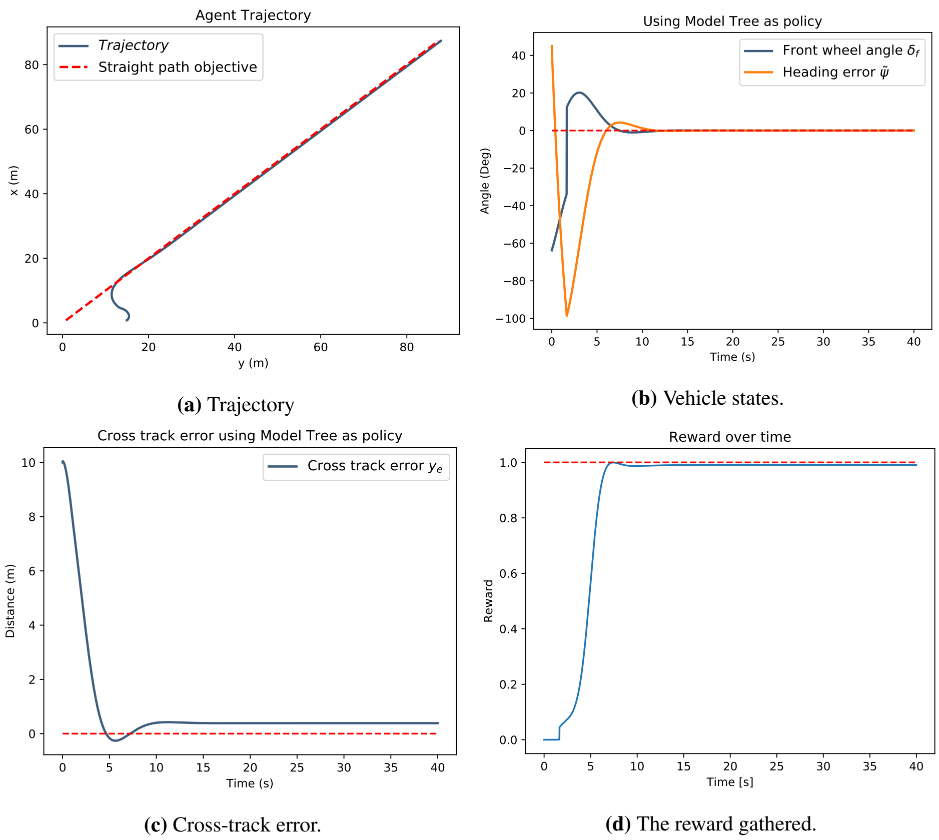
(b) Contour plot of the Model Tree policy

Figure 4.12: Graphical plots of the LMT depth 2

[2]Not to be confused with Logistic Model Tree, a classification type model tree.

Performance

Next, the DDPG agent is replaced by the model tree. Using the same initial conditions as the DDPG agent in Section 4.4 the following simulations seen in Figure 4.13 are made. Comparing with the DDPG agent, the model tree converges to the path though with a larger steady state error, seen in Figure 4.13c. Observing the controller input u in Figure 4.13b it is clear that the model tree operates with a similar policy as the DDPG agent. The difference being that the LMT depth 2 have a larger overshoot in the commanded front wheel angle δ_f in Figure 4.13b.



(a) Trajectory

(b) Vehicle states.

(c) Cross-track error.

(d) The reward gathered.

Figure 4.13: Plots of the simulation when using LMT depth 2 as the agent when the vehicle is initialised with $y_e = 10$ and $\tilde{\psi} = \frac{\pi}{4}$ rad.

4.5.2 Model Tree depth 3

Further increasing the depth for the model tree increases the number of partitions in the input space of the policy. This could result in better approximations of the DDPG policy. A LMT of depth 3 is trained and results in the tree visualised in Figure 4.14. An enlarged figure is shown in the appendix Figure A.1. The number of leaf nodes grows exponentially with the depth (2^d) and for a LMT of depth 3, this results in 8 leaf nodes. Next, the policy is plotted in Figure 4.15. Comparing the plots with the policy of LMT depth 2 it is clear that the LMT depth 3 approximate the shape of the DDPG policy better. This is a direct result of increasing the depth and further allowing the tree to split the input into finer regions. Another apparent improvement is the reduced areas of the policy outside the bounds of [-1, 1]. Finally, the discontinuous jumps in between the split regions of the tree are smaller compared to the lower depth LMT. This is desirable and a consequence of an improved approximation of the DDPG agent. The performance of the deeper LMT is further analysed in Section 4.6.

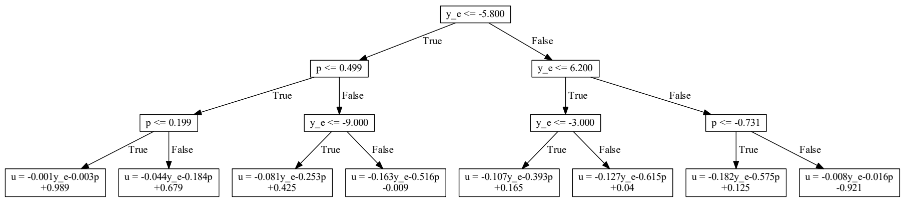
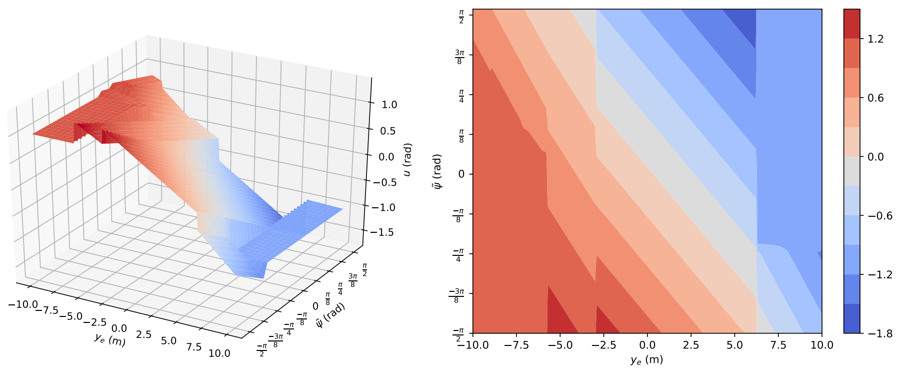

y_e <= -5.800
True
False
p <= 0.499
y_e <= 6.200
True
False
True
False
p <= 0.199
y_e <= -9.000
y_e <= -3.000
p <= -0.731
True
False
True
False
True
False
True
False
u = -0.001y_e-0.003p +0.989
u = -0.044y_e-0.184p +0.679
u = -0.081y_e-0.253p +0.425
u = -0.163y_e-0.516p -0.009
u = -0.107y_e-0.393p +0.165
u = -0.127y_e-0.615p +0.04
u = -0.182y_e-0.575p +0.125
u = -0.008y_e-0.016p -0.921

Figure 4.14: The Linear Model Tree with depth 3



(a) Surface plot of the Model Tree policy

(b) Contour plot of the Model Tree policy

Figure 4.15

4.5.3 Quadratic Model Tree

Because of the nature of Linear Model Trees, some discontinuous jumps are observed between split regions in the policy. Another model tree with a quadratic function at the leaf nodes is trained to investigate whether it produces smoother transitions between the regions and increase similarity to the DDPG agent[3]. Employing the training strategy in-

[3]Higher order polynomials are not included since they lack the interpretability observed in linear functions. Secondly, the number of coefficients needed for each function expression grows exponentially with the degree of

troduced in Section 4.5, a Quadratic Model Tree of depth 2 is trained and the structure is shown in Figure 4.16. As seen in the leaf nodes, the number of coefficients are increased significantly when comparing them to the Linear Model Trees.

Observing the policy plots in Figure 4.17 it is seen that the jumps between the split regions are smaller compared to the previous LMT depth 2 and LMT depth 3. Looking at the surface plot, the transition appears almost smooth. Next, since the splits are performed along the vertical and horizontal axis, the LMT struggles to capture curvature along the diagonal. The quadratic model tree on the other hand manages to conform to the diagonal curvature because of the nonlinear terms. As a result, the quadratic model tree depth 2 appears to approximate the DDPG better than LMT depth 2 and depth 3. The performance of deeper trees could be improved and is further investigated in the next section.

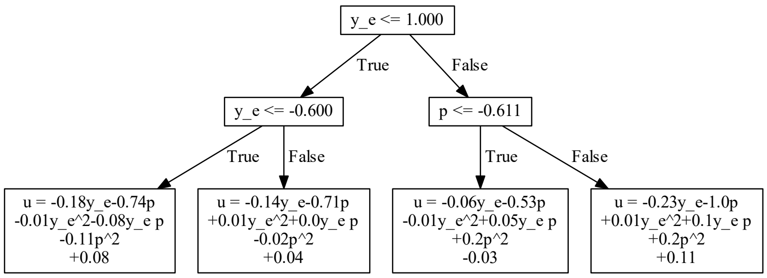
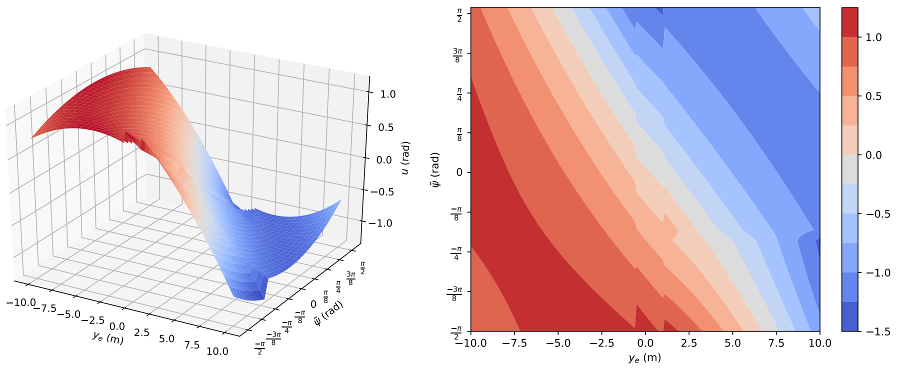

y_e <= 1.000
True
False
y_e <= -0.600
p <= -0.611
True
False
True
False
u = -0.18y_e-0.74p -0.01y_e^2-0.08y_e p -0.11p^2 +0.08
u = -0.14y_e-0.71p +0.01y_e^2+0.0y_e p -0.02p^2 +0.04
u = -0.06y_e-0.53p -0.01y_e^2+0.05y_e p +0.2p^2 -0.03
u = -0.23y_e-1.0p +0.01y_e^2+0.1y_e p +0.2p^2 +0.11

Figure 4.16: Quadratic Model Tree with depth 2



(a) Surface plot of the policy

(b) Contour plot of the policy

Figure 4.17: Graphical plots of the Quadratic Model Tree depth 2 policy

the polynomial. Quadratic Model Trees are investigated since they could show interesting results when used as a means of comparison.

4.6 Performance comparison

The DDPG agent and LMT depth 2 agent in the previous sections converges and follows the straight path with some error margin. Deeper model trees are trained and tested along with the previously introduced trees for comparison of their performance. The limit to how deep the trees can be grown is dependant on the max depth, improvement criterion and the lower bound of points used to create a regression expression at the leaf nodes. These are selected in the WORTHSPLITTING function in Algorithm 1. There are few reasons to grow the trees unreasonably deep if the performance does not improve. Too deep trees would considerably reduce their readability and be impractical as a means for explainability. Along with deeper LMTs are deeper Quadratic Model Trees created. Model trees of higher order polynomials are not added as they require significantly more nonlinear terms in their leaf nodes and as a result reduce the simplicity of the output.

Simulating each agent over the same 100 random vehicle initialisations reveals their performance towards a cross-track error of zero and their similarity to the DDPG agent which they aim to replicate. The results are shown in Table 4.1 where the best results are highlighted in bold. Each of the columns represents important data used to evaluate the agents. The *average absolute steady state error* is measured by the average absolute value at the last timestep of each simulation. The DDPG agent achieves the best result on the task of lowest possible *average absolute steady state error* over the simulations seen in the table. This is not surprising as the model tree based agents are trained to reproduce the behaviour of the DDPG agent with a traceable and transparent policy. The model trees are sub-optimal approximations and therefore don't replicate the DDPG agent perfectly. An interesting observation is that the Quadratic model trees depth 3, 4 and 5 perform worse on *average absolute steady state error* than quadratic model tree depth 2. However, the performance is improved when it is allowed to grow to a depth of 8. A possible explanation for this is that the quadratic model trees depth 3, 4 and 5 try to capture the overshooting of the DDPG policy, but are not able to do this accurately enough resulting in poor performance. When the depth finally reaches 8, it captures the parts of the DDPG policy that accounts for the overshooting and converges with a lower steady state error. This can be seen in Figure 4.18 with the exception of Figure 4.18a where quadratic model tree depth 5 performs best out of the quadratic model trees.

For the purpose of explicitly evaluating the different model trees' similarity to the DDPG agent, two measures are introduced. The first is *Average DDPG deviating absolute error*, $\overline{|y_e - y_{e_{DDPG}}|}$, found by the formula $\frac{1}{T}\sum_{i=0}^{T}|y_{e_i} - y_{e_{DDPG_i}}|$ where T is the total number of timesteps in 100 runs. The second measure is *Average DDPG deviating steady state error*: $\overline{|y_{e_\infty} - y_{e_{DDPG_\infty}}|}$ calculated by $\frac{1}{N}\sum_{i=0}^{N}|y_{e_\infty} - y_{e_{DDPG_\infty}}|$ where N is the number of simulated runs (100). An ideal replication of the DDPG agent would result in $0\,\mathrm{m}$ on both of these measures. The LMT depth 100* [4] achieves the closest value to zero for these two measures in the simulations. This model tree outperforms the other trees by a large margin on both of the measures. To further investigate its performance, three examples of

[4]This model tree was trained with a lower bound of minimum 10 samples at the leaf nodes instead of the bound 100 which is used for the other trees. This allows the agent to conform to a lower number of samples at the leaf nodes with an increased risk of overfitting. This might not be a problem since the task is to approximate the black box.

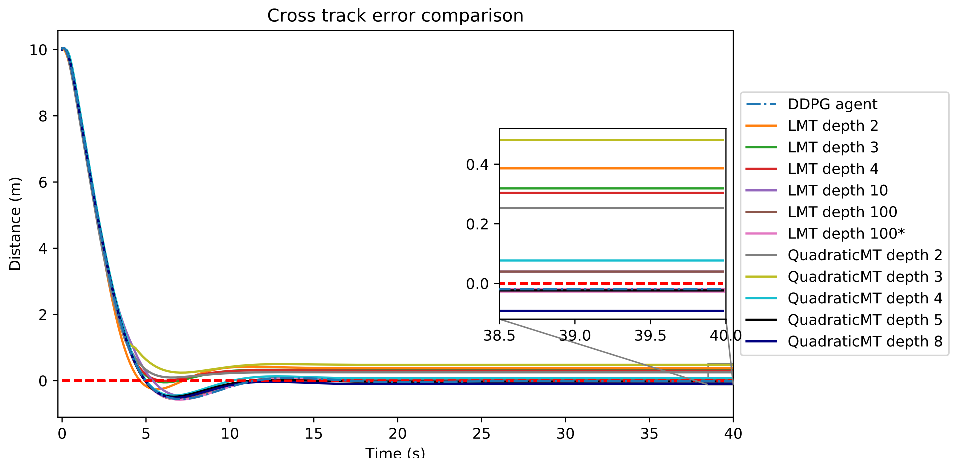
the cross-track error convergence is shown in Figure 4.18. From the figures it is seen that the LMT depth 100*, in pink, approximates the dashed dot blue line of the DDPG agent the best. From these results it is safe to conclude that this model tree achieves the closest approximation of the DDPG agent. Even though it is allowed to grow to a depth of 100, it only reaches a depth of 23 while LMT depth 100 reaches 13. QuadraticMT depth 8 also performs well when looking at the DDPG deviating measures in the table. It is however beaten by LMT depth 10 and there is therefore little incentive to use a nonlinear model at the leaf nodes when a Linear Model Tree with 2 more in depth performs better and is simpler to interpret. A visualisation of LMT depth 10 is shown in the appendix Figure A.2.

Table 4.1: Simulating the agents with 100 random initialisations with $-10 \leq y_e \leq 10$ and $-\frac{\pi}{2} \leq \tilde{\psi} \leq \frac{\pi}{2}$ for 2000 timesteps ($40s$).

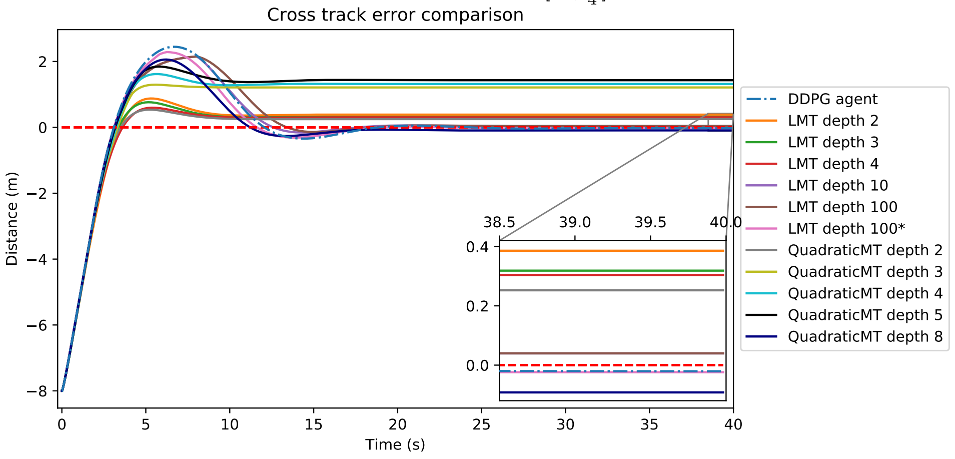
Name	Average absolute steady state error $\overline{\lvert y_{e_\infty} \rvert}$ [m]	Average DDPG deviating absolute error $\overline{\lvert y_e - y_{e_{DDPG}} \rvert}$ [m]	Average DDPG deviating steady state error $\overline{\lvert y_{e_\infty} - y_{e_{DDPG_\infty}} \rvert}$ [m]
DDPG agent	***0.0202***	*0.0000*	*0.0000*
LMT depth 2	0.4469	0.5439	0.4671
LMT depth 3	0.3188	0.4070	0.3390
LMT depth 4	0.3039	0.4064	0.3241
LMT depth 10	0.0397	0.0905	0.0599
LMT depth 100	0.0397	0.0953	0.0599
LMT depth 100*	0.0238	**0.0388**	**0.0036**
QuadraticMT depth 2	0.2524	0.3738	0.2727
QuadraticMT depth 3	0.9925	0.9167	1.0127
QuadraticMT depth 4	0.7094	0.6248	0.7296
QuadraticMT depth 5	0.7419	0.6265	0.7427
QuadraticMT depth 8	0.1853	0.1870	0.1666

Out of the simpler LMTs, LMT depth 2, 3 and 4 are considered as they have less depth and are piecewise linear ensuring simplicity in their interpretation. LMT depth 2 achieves an average DDPG deviating steady state error of $0.4671\,\mathrm{m}$ while LMT depth 3 and LMT depth 4 average at $0.3390\,\mathrm{m}$ and $0.3241\,\mathrm{m}$, respectively. This shows that there's an average reduction in DDPG deviating steady state error by 27.4% when increasing the depth from 2 to 3 for these agents. Increasing the depth from 2 to 4 reduces the deviation by 30.6% but significantly increases the number of leaf nodes in the LMT. As a trade-off between performance and explainability, a LMT of depth 3 is selected for further analysis in Section 4.7.2.

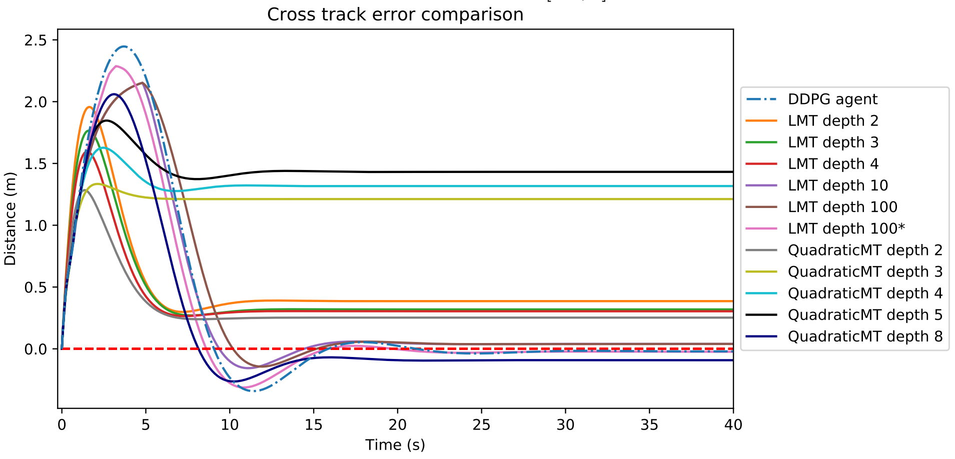
From the analysis of the performance of the model trees and the DDPG it is clear that the model tree does not replicate the the DDPG agent in a global optimal way, but sub-optimal. A possible solution to improve the approximation is to increase the amounts data used in the training set for the model trees, allow infinite depths and remove any constraint of minimum samples for each leaf node. The immediate drawback is longer training time and it could quickly become infeasible to train because of the *curse of dimensionality*. This solution could ironically lead to uninterpretable trees as they become unreasonably large and result in a black box system. With LMTs it is therefore necessary to give up some accuracy in the approximation to allow for transparent and interpretable trees with a reasonable depth.



(a) Initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$



(b) Initialisation $\mathbf{x}_0 = [-8, 0]$



(c) Initialisation $\mathbf{x}_0 = [0, \frac{\pi}{2}]$

Figure 4.18: The cross-track error from three simulations.
The model trees aim to replicate the DDPG agent plotted in blue dashed-dot.

4.7 Explaining the DDPG agent

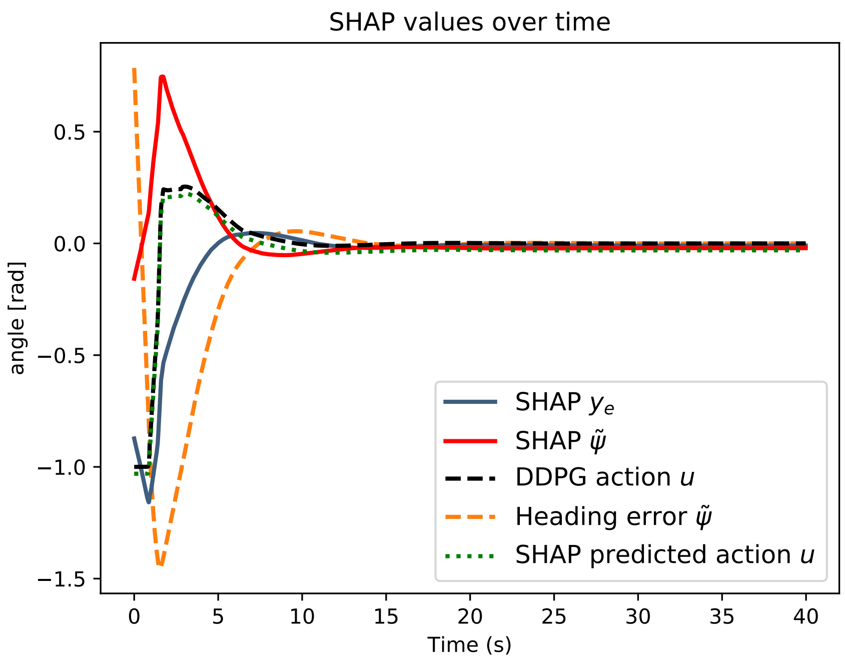
The DDPG agent trained in Section 4.4 operates in a black box manner when computing the input-output relationship between the state $\mathbf{x} = [y_e, \tilde{\psi}]$ and the control action $u = \pi(\mathbf{x})$. This is simply because the function expression for the DDPG is dependant on the many internal weights and activation functions in the neural network. In an effort to explain the DDPG agent, two techniques are tested, starting with SHAP followed by LMT depth 3. The section finishes with a discussion on how these two methods compare when explaining the DDPG agent.

4.7.1 SHAP explanation

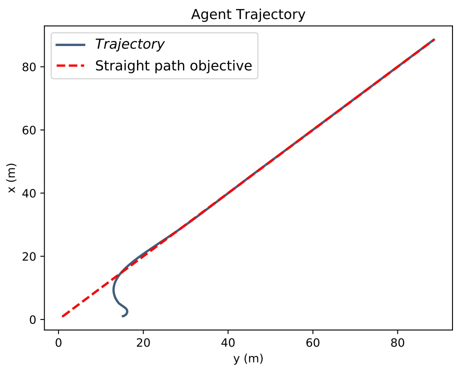
SHAP calculates the contribution of a feature value to the prediction, as introduced in Section 3.4. The SHAP method creates a special weighted linear regression model of the true model. It is based on the game theoretic approach of Shapley values made to ensure a fair, unique distribution of the surplus amongst participating players. The players in the context of this thesis is the states (the input) and the surplus is the consequent action (output).

The SHAP background model is created by using 1681 uniformly distributed samples of the DDPG policy. The background samples are taken in a $41 \cdot 41$ grid within the input space $-10 \leq y_e \leq 10$ and $-\frac{\pi}{2} \leq \tilde{\psi} \leq \frac{\pi}{2}$. The simulation of the DDPG agent with initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$ is used in this section. This is the simulation used in Section 4.4 and plotted in Figure 4.9. The 2000 timesteps results in 2000 input-output observations which is utilised for the SHAP explanation. The data is passed to the SHAP Kernel explainer introduced in Section 3.4.2 to calculate the SHAP values for the input features. The resulting SHAP values over time are shown in Figure 4.19a where the sum of the SHAP values adds up to the SHAP model of the DDPG action u during each timestep. This way of using SHAP values to explain an agent is inspired by the work in [50]. The SHAP predicted values are shown in dotted green. Looking at the plot it is observed that $\tilde{\psi}$ has a low, negative SHAP value during the first second while the cross-track error y_e has a larger negative SHAP value. This could be interpreted as the DDPG agent sees the cross-track error as more important than the heading error during this interval. The agent therefore enforces the control action of maximum turning towards the left ($u = -1$) until the agent reaches a heading error $\tilde{\psi} \approx -1\text{rad}$. During the next half second, as $\tilde{\psi} \rightarrow -\frac{\pi}{2}\text{rad}$, meaning the vehicle heading reaches perpendicular to the objective path, the SHAP value of $\tilde{\psi}$ increases towards 1. This means that the state now contributes to turning the wheel towards the right, in the opposite direction. The SHAP value of the cross-track error is still negative, but its magnitude decreases. This results in the DDPG agent reducing its negative angle, finally turning slightly positive between about $2s < t < 10s$ so the vehicle heading aligns with the path over time, i.e. $\tilde{\psi} \rightarrow 0\text{rad}$. Once the control action to the vehicle has converged around 14s, the SHAP values remain approximately constant, cancelling each other out.

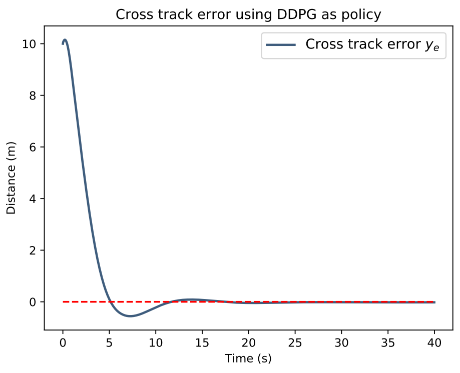
Using the SHAP values to explain the simulated run hints to a strategy of turning the wheel such that the vehicle is perpendicular to the path when the cross-track error is high and the heading error is low in magnitude, relative to each other. As the vehicle reaches a perpendicular heading to the path, $\tilde{\psi} \rightarrow \pm\frac{\pi}{2}\text{rad}$, and the cross-track error is decreasing in



(a) The SHAP values plotted over time. The SHAP values sums up, at each time instance, to the SHAP model of the DDPG action u during each timestep. The DDPG action u in dashed black. The y-axis represents angle value in rad.



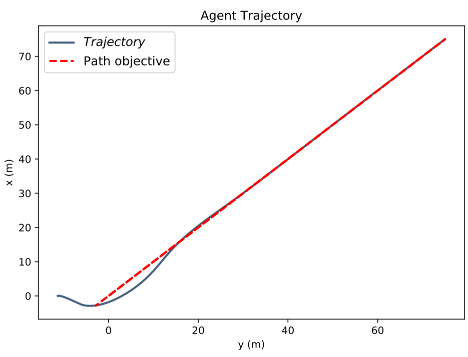
(b) The trajectory.



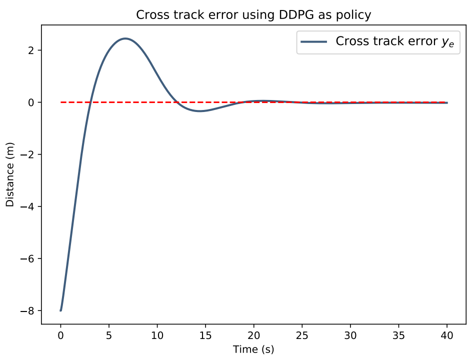
(c) The cross-track error.

Figure 4.19: The SHAP values, heading error $\tilde{\psi}$, cross-track error and trajectory y_e plotted over time for the DDPG agent controlling the vehicle with initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$.

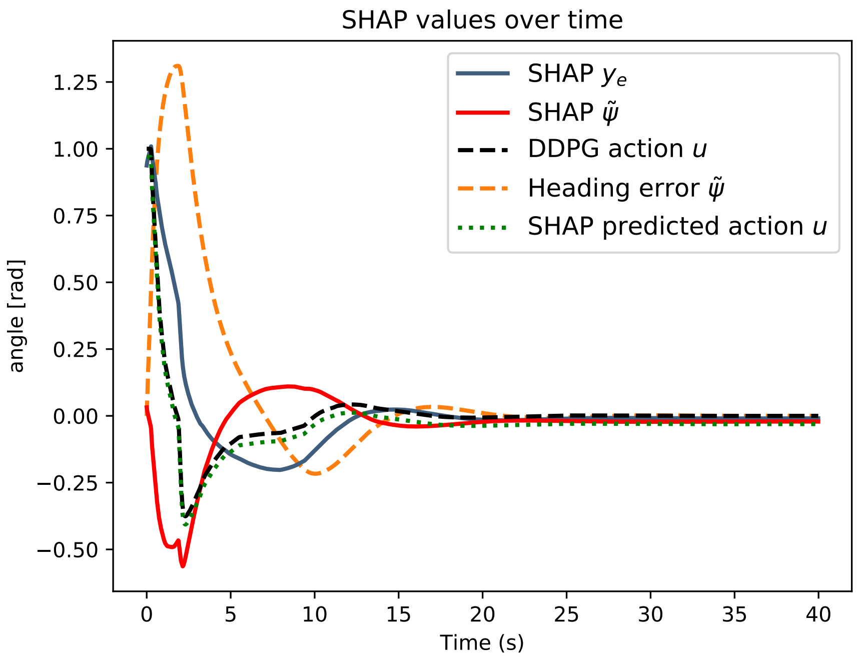
magnitude, the DDPG agent straighten up the vehicle by turning the wheel in the opposite direction. Finally, the vehicle keeps the wheel at zero angle until the run is over. This hypothesis can be tested by performing further simulated runs. As such, another scenario is examined with the SHAP method. Starting the vehicle in $\mathbf{x}_0 = [-8, 0]$, the trajectory, cross-track error and SHAP values are visualised in Figure 4.20. As seen in Figure 4.20c, the SHAP value of the cross-track error starts close to 1, contributing to the DDPG action to turn towards the right, while the SHAP value of $\tilde{\psi}$ is close to zero. The cross-track error is large compared to the heading error which starts around zero. As the vehicle reaches a perpendicular heading, the SHAP value of $\tilde{\psi}$ becomes large negative while the SHAP value of y_e decreases in magnitude towards zero. These contributions result in the DDPG agent steering the front wheel towards the opposite direction, to the left with $u \approx -0.4$. This steadily straightens up the vehicle until the cross-track error converges in an under-damped manner towards the goal of $y_e = 0$. This strengthens the hypothesis of the strategy suspected from the previous simulation. The SHAP values shows valuable insights into the black box DDPG agent, but are suggestions and not complete explanations. The agent still acts in an opaque manner and a different method is needed to completely decipher the internal workings of the DDPG agent.



(a) The trajectory.



(b) The cross-track error y_e



(c) SHAP values

Figure 4.20: The SHAP values, heading error $\tilde{\psi}$ and cross-track error y_e plotted over time for the DDPG agent controlling the vehicle with initialisation $\mathbf{x}_0 = [-8, 0]$.

4.7.2 Using Linear Model Tree as explanation

Since the Linear Model Tree uses linear leaf nodes to compute its output, the coefficients can be interpreted as weights scaling the states' impact to the control action. This is expressed as

$$u_{LMT} = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \tag{4.15}$$

where w_i is the weight to state x_i. Since the bias term w_0 does not change as the states vary, its contribution is averaged amongst the weight-state pairs such that state number i contributes

$$\phi_i = w_ix_i + \frac{w_0}{n}, \forall i > 0, n > 0. \tag{4.16}$$

Summing these contributions results in

$$u_{LMT}(\mathbf{x}) = \phi_1 + \cdots + \phi_i + \cdots + \phi_n \approx u_{DDPG}(\mathbf{x}). \tag{4.17}$$

From Section 4.6 LMT depth 3 is found to approximate the DDPG fairly well while avoiding being very deep compared to the other deeper trees. The vehicle is simulated using the DDPG agent and the initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$ as performed in the previous section, Section 4.7.1. Simultaneously, the LMT depth 3 computes its control action u_{LMT} using the same states. The weighted contributions of the LMT depth 3 are then plotted with the DDPG agent in Figure 4.21 to show how well the LMT approximates the DDPG during this simulation. The contributions in blue and red sums up, at each time instance, to the LMT predicted control input u_{LMT} (in dotted green) which approximately predicts the DDPG agent's action (in dashed black). Both contributions have values around -0.5 at the interval $t = 0s$ to $t \approx 0.5s$ which sum to ≈ -1. As the heading error $\tilde{\psi}$ goes towards -1.5rad around $t = 1s$, the model tree changes its leaf function and consequently, each states' contribution. The DDPG action (and LMT predicted action) goes from $u = -1$ to $u = 0.25$ during the next $0.5s$ since the contributions of the $\tilde{\psi}$ spikes and influences the agent to turn towards the right to reduce the heading error $\tilde{\psi}$. The vehicle converges over time with a similar reasoning behind the state contributions as in SHAP. Notice however that the green dotted line, the LMT predicted action u, is slightly offset from the actual DDPG action u (in dashed black). This is simply because the LMT approximates the DDPG agent sub-optimally.

In order to understand which leaf nodes the model tree uses during a simulation, a tool is created and visualised in Figure 4.22. The tool is helpful to estimate the importance of each leaf in the model tree control policy. The leaf with policy $u = -0.127y_e - 0.615\tilde{\psi} + 0.04$ is the most used during the convergence of the vehicle. The policy in this region could be of more significance since it guides the vehicle towards the desired state of $y_e = 0$. Looking back at the tree structure in Figure 4.14, this leaf is responsible for the policy when $-3.000 \leq y_e \leq 6.200$. It could therefore be of interest to weight the policy to be more accurate in this region compared to the edge regions as the goal finally is to reach the same y_e as the DDPG agent.

4.7.3 Comparing explanations by SHAP and LMT

One of the major advantages of the SHAP explanation is their underlying theory of Shapley values obeying the four desirable properties: symmetry, efficiency, law of aggregation and null player. The input features are a fairly and uniquely distributed among the features. It

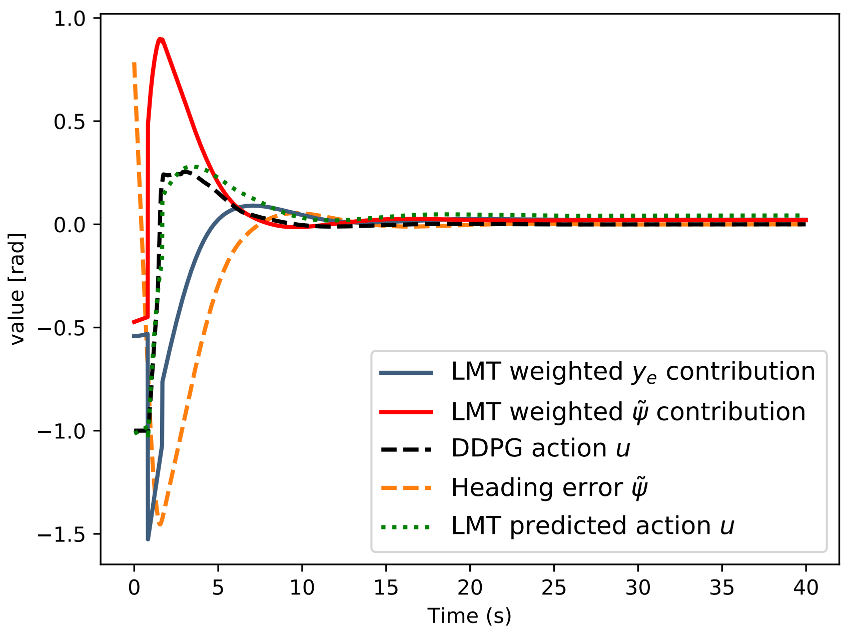


Figure 4.21: The weighted contributions of each state over time when using LMT depth 3 on the initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$.

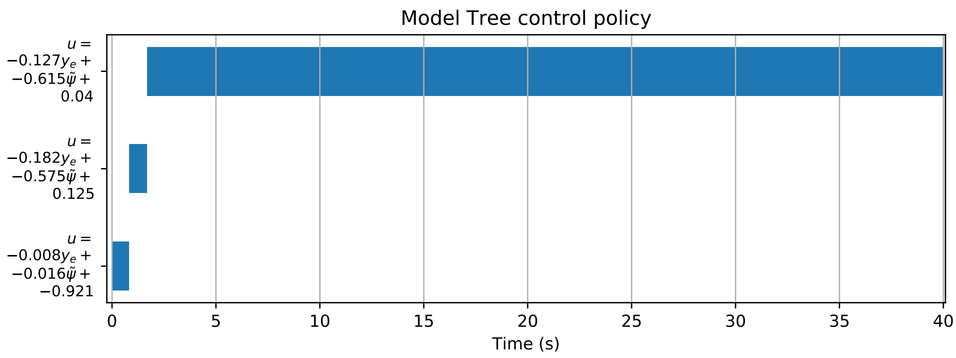


Figure 4.22: A visualisation of leaf nodes visited during the simulation of the model tree controlling the vehicle.

is also a model-agnostic explanation method which means that it is applicable to any black box system, whether the inputs are categorical or numerical and the output is regression or classification. This is not the case for LMTs which in this thesis is applied on a continuous input-output regression.

Based on the gathered information by decomposing the states into SHAP values a few takeaways can be made. First, the importance of the states changes over time depending on their magnitude. This can be used as a model to explain how the features influences the black box agents actions. However, using SHAP values to explain the agent is not enough to reveal the inner structure of the agent. SHAP creates a special weighted linear approximation of the DDPG. This means that the approximation model is still not a guaranteed accurate approximation of the underlying DDPG policy. For example, in Section 4.7.1 it is suggested, based on the observed SHAP values, that the DDPG agent chooses a strategy of reaching the objective path in a shortest amount of time. This is done by instantly turn its heading perpendicular to the path. As the cross-track error goes towards zero, the agent turns the vehicle to align with the path. This strategy makes intuitive sense, but there is no way of proving it using SHAP and it remains a hypothesis. This means that using SHAP can be a helpful as a tool to analyse behaviour, but it does not, to the best of the author's knowledge, provide a transparent policy.

The LMT explanation with the weighted contributions and visualisation of leaf nodes visited are helpful to analyse the behaviour of the LMT and DDPG agent. Combining the results from Figure 4.19a and Figure 4.21 results in Figure 4.23. As seen in the figure, the contributions decomposed by the LMT shows a similarity to the SHAP values. Based on the few examples provided, this suggests that the LMT captures similar relationships between the input features like SHAP. The difference being that the weights are known on a global level in the model tree visualisation, which SHAP does not provide. The LMT weighted contribution have a large benefit of being computationally inexpensive when creating an explanation compared to SHAP. However, it does require a trained model tree which may be computationally expensive in the first place.

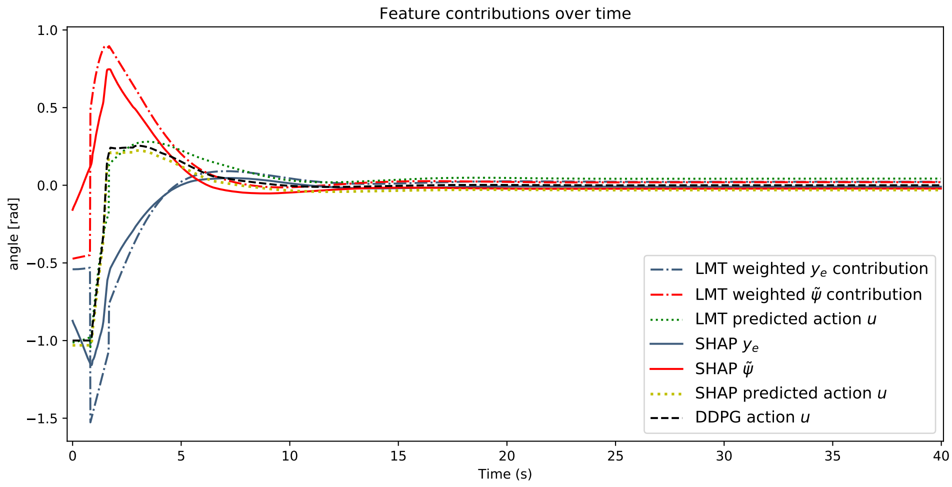


Figure 4.23: The SHAP and LMT explanations combined in one figure for the simulation with initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$.

If an expert is satisfied with the performance of a certain LMT over many test cases, then it may replace the opaque AI driven control algorithm. By employing the fully transparent

LMT policy the expert essentially replaces the black box with a transparent box. The control policy can be traced from the input root, through the split conditions and down to the output leaf node. However, the new box is still an approximation, meaning that the performance is slightly different than the original algorithm. Depending on the extent of declined performance, the upside of having a transparent algorithm could outweigh a black box model because of safety and explainability concerns.

Trees as an explanation technique conforms to the findings by Miller et al. [38], presented pointwise in Section 3.2. It explains why a leaf policy happens over another leaf policy, which is point number 1: *Explanations should explain why event A happened instead of B*. Point number 4 reads: *An explanation is a conversational transfer of knowledge presented in belief of the recipient's view*. It is worth noting that this point suggest that a tree with less leaf nodes are preferable to a tree with more leaf nodes. Since the model tree is a visualisation, it is a conversational transfer of knowledge. As the amount of nodes increase with the depth, the recipient needs to take more leaf policies into account when interpreting the model on a global level. The depth should therefore be as low as possible, while maintaining sufficient accuracy.

The interesting takeaway from this experiment is the power of model trees to capture the underlying policy, performing almost as well as the original policy while allowing a transparent policy. Increasing depth seems to improve the approximation but reduces the simplicity of the tree. A balance between depth and simplicity is needed.

4.8 Applying agents on curved path following

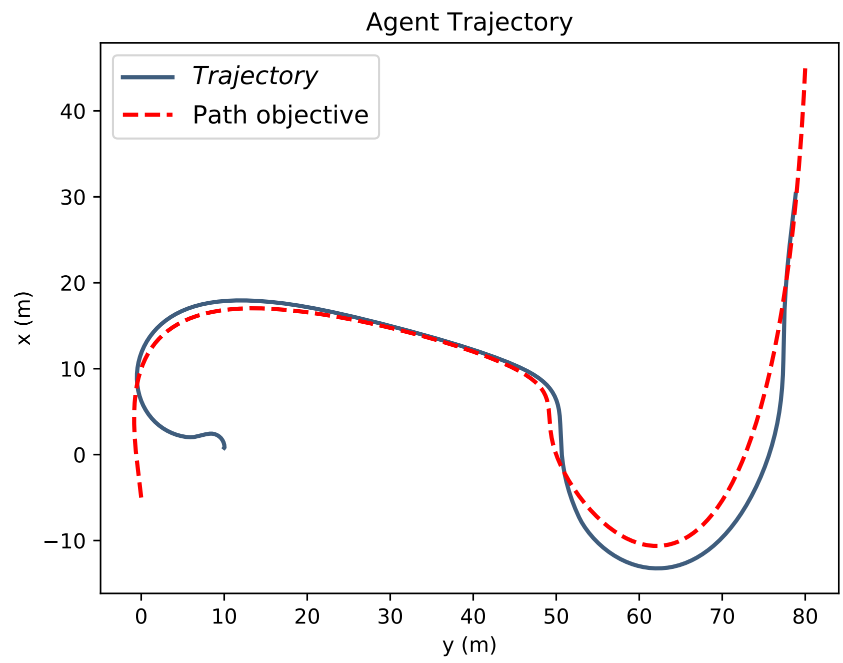
The DDPG agent and LMT depth 3 is tested on a curved path without additional training to investigate their performance on a more challenging task. Since the agents still use limited information of only the cross-track error and heading error, $\mathbf{x} = [y_e, \tilde{\psi}]$, a slowly varying curve is created. This ensures that the path angle, $\alpha_p(\omega)$, does not change value too rapidly. The path is created by interpolating a third order spline using the waypoints in Table 4.2. This way of creating a curved path is inspired by [12] and [34].

Table 4.2: Table over parameters used to generate a curved path

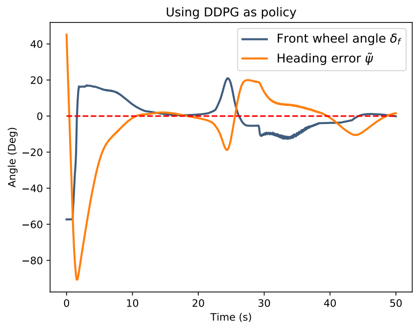
Waypoint	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Path parameter ω	0	1	2	3	4	5	6
x	-5	10	17	10	0	-10	45
y	0	0	15	45	50	65	80

The agents are simulated for 2500 timesteps resulting in a total time of $t = 50s$ where the initialisation is $\mathbf{x}_0 = [10, \frac{\pi}{4}]$. The DDPG agent is plotted in Figure 4.24 where it is observed that the vehicle converges. The agent does not converge during the curves but manages to reduce the cross-track error when the path straightens out. Even though the agent has limited information about the environment it is still able to reach the targeted goal of zero cross-track error with a slight margin.

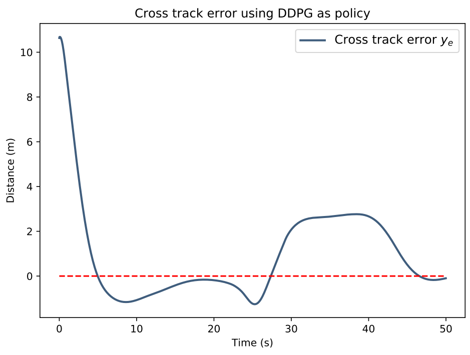
Looking at Figure 4.25, it is seen that the LMT depth 3 manages to converge to the path



(a) The trajectory.



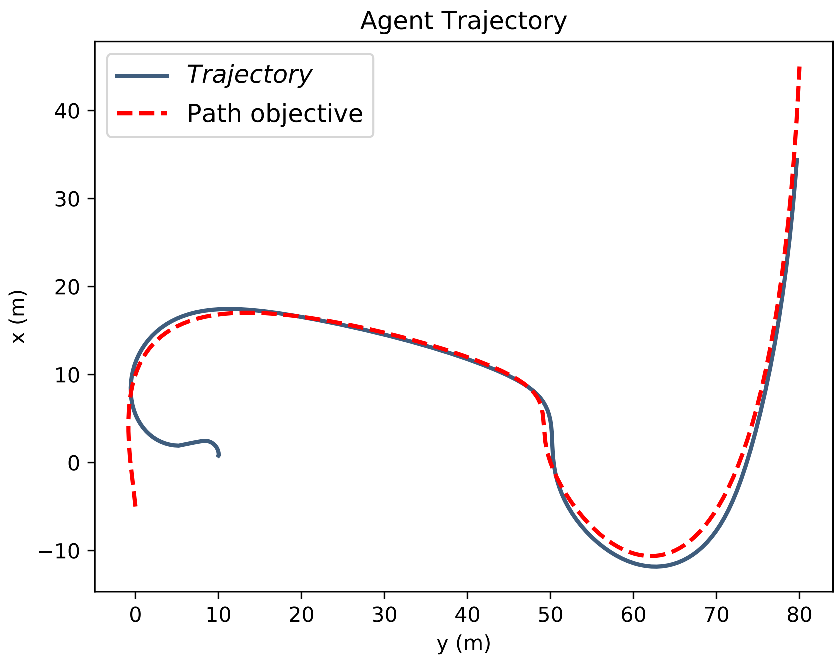
(b) Controller input and heading error, $\tilde{\psi}$.



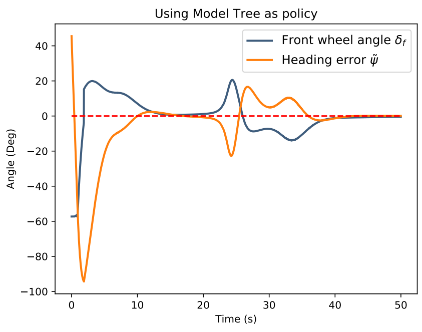
(c) The cross-track error y_e

Figure 4.24: The simulation for the DDPG agent controlling the vehicle with initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$ on a curved path.

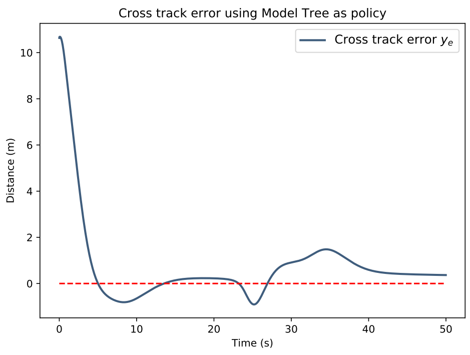
with some steady state error. Even though the LMT is trained on samples from the DDPG agent which was trained on a straight-line path following task, it is able to reach the objective with some margin. The task of curved path following is a significantly more challenging problem than straight path because of the varying curvature. This further suggest that the use of Linear Model Trees is a viable strategy to transform DRL-based black box policies into transparent and explainable policies.



(a) The trajectory.



(b) Controller input and heading error



(c) The cross-track error y_e

Figure 4.25: The simulation for the LMT depth 3 agent controlling the vehicle with initialisation $\mathbf{x}_0 = [10, \frac{\pi}{4}]$ on a curved path.

4.9 Recommendations for training Model Trees to approximate black box DRL agents

Other works using model trees have mentioned the benefit of using this method since it results in transparent rules [10]. Though, no works have been found to use them to create a transparent, rule based policy in a DRL setting. The closest found being Soft Decision Trees (SDT) discussed in Chapter 1 which is used on a discrete action space. The resulting SDT is not able to produce rule based explanations, which LMTs are able to. As such, a series of recommendations based on researched literature and experiences are made to clarify how to transform a DRL-policy into a LMT.

- As noted in Section 4.5, sampling of the black box agent needs to be performed uniformly within the targeted input state space. A lower and upper bound for each state have to be selected on which the sampling points will be distributed uniformly. In this thesis, the upper and lower bounds were selected as the same as the maximum and minimum states in the initialisation of the vehicle during training. This ensures that the LMT learns from all regions of the DRL policy in which it is designed to operate. The sampling is also performed in a grid with $100 \cdot 104$ points, aiming to keep a balanced sampling across the input space.
- From the uniformly distributed sampling points, use the input, $\mathbf{x}_i$, and consequent output policy action $u_i = \pi(\mathbf{x}_i)$ as training data for the model tree.
- Select a depth which provides a trade off between accuracy and transparency. Deeper could result in better approximations, but results in a larger number of leaf nodes which reduces interpretability.
- Select a constraint of samples as the minimum a leaf node may have to ensure generalisation. In this thesis this was set mainly to 100.
- Linear leaf nodes are preferred over quadratic or other nonlinear functions because of simplicity and interpretability. As such, a linear function at the leaf node is suggested.
- Once LMT has finished training, replace the black box agent with the LMT and investigate the performance. If the behaviour is poor it may be a sign of a policy which needs a deeper LMT to accurately approximate it.

4.10 Remark

For the vehicle model, the assumption of the velocity vector pointing in the direction of the wheel is no longer accurate at higher velocities. Under such circumstances, a dynamic model for the lateral vehicle motion needs to be considered instead of a kinematic model. A suitable model for this purpose is described in [21] where the authors develop a linear and non-linear observer for the side-slip angles.

Chapter 5

Conclusion

5.1 Conclusion

Explainable AI is an emerging field gaining attention in a world increasingly driven by AI applications. The field comes as a response to the concerns of employing opaque models in systems without understanding their internal workings. Linear Model Trees are suggested as an approach to transform a black box policy into a piecewise linear policy, making it traceable and transparent from input to output. The results showed they were able to capture the underlying policy and may replace a Deep Reinforcement Learning-agent. This comes with a minor deterioration of performance in the experimented environment. Trees allowed to grow deeper showed improved approximation to the black box policy. The transparency of the tree is reduced proportionally to the depth and a trade off between accuracy and transparency must be made. The depth should therefore be as low as possible, while maintaining sufficient accuracy. Using a quadratic model tree performed well when the depth was 2 compared to a LMT of depth 2, but the simplicity of linear functions is lost and quadratic model tree was consequently discarded as a transparent approach.

SHAP was used as a means to explain the black box DDPG policy. The method provided a suggestion to how the features contributed to the output, and a hypothesis of the internal workings was made. It was helpful as a tool to analyse the behaviour, but it did not provide a transparent global policy of the agent. Model trees, on the other hand, approximates the agent with a global policy. The coefficients in the linear models in LMTs can be interpreted as weights scaling the states' impact to the control action. Like SHAP, these contributions can be used to explain the states' importance to the DDPG. It was noted that the sum of contributions explained by the LMT depth 3 were slightly offset from the DDPG since the LMT has slightly degraded performance when predicting the DDPG policy. The LMT feature contributions did at the same time show the actual importance to the LMT agent's output. As such, if an expert is satisfied with the performance of the LMT of a certain depth, then it may replace an opaque DRL control algorithm. A tool for visualising the leaf nodes of the model tree was created which could help an expert to gain understanding of how the agent behaves during a simulation. Finally, a series of recommendations have been proposed to guide training of model trees to approximate black box DRL agents. To the author's best knowledge, no work has used Linear Model Trees as a method to transform a black box policy into a transparent policy. Consequently, Linear Model Trees deserve further attention in the XAI research community and could be a step in the direction towards new tools for creating transparent DRL agents.

5.2 Further work

The approach of using model trees to transform a black box policy into a transparent policy opens up many possible strategies for additional findings. The first being to apply model trees on more complicated systems to further investigate their performance and applicable domains. This could be systems with a higher degree of nonlinearities where the consequent DRL policy is more difficult to approximate. This could also be combined with experiments on a physical system to test the application in a non-simulated environment, for instance robotic systems or a quadcopter.

Further investigation into the selections of parameters for training a model tree could guide towards new recommendations. For instance, adding a constant in front of Δ-function could result in model trees allowed to grow deeper only when the improvement is significant. This could work as a tradeoff parameter deciding between accuracy and interpretability through the depth. Moreover, investigating trees which adapts to desired accuracy and interpretability could be another interesting take on the tradeoff problem.

As an improvement, a promising next step could be to experiment with a different type of split condition. The model trees used in this thesis have axis parallel splits and could achieve improvement by allowing oblique splits, see for instance [64]. These splits could allow for a greater fit of the regression models at the leaf nodes.

Bibliography

[1] Adadi, A., Berrada, M., 2018. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access 6, 52138–52160.

[2] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W., 2015. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. PLOS ONE 10, 1–46. URL: https://doi.org/10.1371/journal.pone.0130140, doi:10.1371/journal.pone.0130140. publisher: Public Library of Science.

[3] Breiman, L., 2001. Random Forests. Machine Learning 45, 5–32. URL: https://doi.org/10.1023/A:1010933404324, doi:10.1023/A:1010933404324.

[4] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A., 1984. Classification and regression trees. CRC press.

[5] Carbone, N.B., 2019. An overview and comparison of Explainable AI (XAI) methods. Specialization Project TTK4550. Norwegian University of Science and Technology, Faculty of Information Technology and Electrical Engineering, Department of Engineering Cybernetics.

[6] Clinciu, M.A., Hastie, H., 2019. A Survey of Explainable AI Terminology, in: Proceedings of the 1st Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence (NL4XAI 2019), pp. 8–13.

[7] Coppens, Y., Efthymiadis, K., Lenaerts, T., Nowe, A., 2019. Distilling Deep Reinforcement Learning Policies in Soft Decision Trees, in: Miller, T., Weber, R., Magazzeni, D. (Eds.), Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence, pp. 1–6.

[8] Danwei Wang, Feng Qi, 2001. Trajectory planning for a four-wheel-steering vehicle, in: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), pp. 3320–3325 vol.4.

[9] Deng, J., Yang, Z., Li, Y., Samaras, D., Wang, F., 2020. Towards Better Opioid Antagonists Using Deep Reinforcement Learning. _eprint: 2004.04768.

[10] Etemad-Shahidi, A., Mahjoobi, J., 2009. Comparison between M5 model tree and neural networks for prediction of significant wave height in Lake Superior. Ocean Engineering 36, 1175 – 1181. URL: http://www.sciencedirect.com/science/article/pii/S0029801809001905, doi:https://doi.org/10.1016/j.oceaneng.2009.08.008.

[11] European Parliament, Council of the European Union, 2016. General Data Protection Regulation. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj. [Accessed 2020-05-27].

[12] Fossen, T.I., 2011. Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons.

[13] GitHub, Inc., . GitHub. URL: https://github.com/.

[14] Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.

[15] Google LLC, . Google Colaboratory. URL: https://colab.research.google.com/. https://research.google.com/colaboratory/faq.html.

[16] Grigorescu, S., Trasnea, B., Cocias, T., Macesanu, G., 2020. A survey of deep learning techniques for autonomous driving. Journal of Field Robotics 37, 362–386. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21918, doi:10.1002/rob.21918. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21918.

[17] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., others, 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks. IEEE Press.

[18] Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. Computing in Science & Engineering 9, 90–95. doi:10.1109/MCSE.2007.55. publisher: IEEE COMPUTER SOC.

[19] Jia, R., Dao, D., Wang, B., Hubis, F.A., Hynes, N., Gurel, N.M., Li, B., Zhang, C., Song, D., Spanos, C., 2019. Towards efficient data valuation based on the shapley value. arXiv preprint arXiv:1902.10275 .

[20] Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. Journal of artificial intelligence research 4, 237–285.

[21] Kiencke, U., Daiß, A., 1997. Observation of lateral vehicle dynamics. Control Engineering Practice 5, 1145 – 1150. URL: http://www.sciencedirect.com/science/article/pii/S0967066197001081, doi:https://doi.org/10.1016/S0967-0661(97)00108-1.

[22] Komite Nasional Keselamatan Transportasi, 2019. Aircraft Accident Investigation Report PT. Lion Mentari Airlines Boeing 737-8 (MAX); PK-LQP Tanjung Karawang, West Java Republic of Indonesia 29 October 2018. Technical Report. URL: http://knkt.dephub.go.id/knkt/ntsc_aviation/baru/2018%20-%20035%20-%20PK-LQP%20Final%20Report.pdf. [Accessed 2020-05-27].

[23] Lavecchia, A., 2019. Deep learning in drug discovery: opportunities, challenges and future prospects. Drug Discovery Today 24, 2017 – 2032. URL: http://www.sciencedirect.com/science/article/pii/S135964461930282X, doi:https://doi.org/10.1016/j.drudis.2019.07.006.

[24] Lazo, L., Paul Schemm, Aratani, L., 2019. Investigators find 2nd piece of key evidence in crash of Boeing 737 Max 8 in Ethiopia URL: https://www.washingtonpost.com/world/africa/french-start-analysis-ethiopian-airlines-black-boxes-as-new-evidenc
2019/03/15/87770e8c-468f-11e9-94ab-d2dda3c0df52_story.html. [Accessed 2020-05-27].

[25] Lekkas, A.M., Fossen, T.I., 2012. A time-varying lookahead distance guidance law for path following. IFAC Proceedings Volumes 45, 398–403. Publisher: Elsevier.

[26] LeNail, A., 2019. NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software 4, 747. URL: http://dx.doi.org/10.21105/joss.00747, doi:10.21105/joss.00747. publisher: The Open Journal.

[27] Liessner, R., Schroer, C., Dietermann, A., Bäker, B., 2018. Deep Reinforcement Learning for Advanced Energy Management of Hybrid Electric Vehicles, pp. 61–72. doi:10.5220/0006573000610072.

[28] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. _eprint: 1509.02971.

[29] Liu, G., Schulte, O., Zhu, W., Li, Q., 2019. Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees, in: Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., Ifrim, G. (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, Cham. pp. 414–429.

[30] LLC, G., . Google Drive. URL: https://www.google.com/drive/.

[31] Lundberg, S.M., Erion, G.G., Lee, S.I., 2018. Consistent Individualized Feature Attribution for Tree Ensembles. _eprint: 1802.03888.

[32] Lundberg, S.M., Lee, S.I., . SHAP (SHapley Additive exPlanations). URL: https://github.com/slundberg/shap.

[33] Lundberg, S.M., Lee, S.I., 2017. A Unified Approach to Interpreting Model Predictions, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[34] Martinsen, A.B., 2018. End-to-end training for path following and control of marine vehicles. Master's thesis. NTNU.

[35] Martinsen, A.B., Lekkas, A.M., 2018a. Curved Path Following with Deep Reinforcement Learning: Results from Three Vessel Models, in: OCEANS 2018 MTS/IEEE Charleston, pp. 1–8.

[36] Martinsen, A.B., Lekkas, A.M., 2018b. Straight-path following for underactuated marine vessels using deep reinforcement learning. IFAC-PapersOnLine 51, 329–334. Publisher: Elsevier.

[37] Mataric, M.J., 1994. Reward Functions for Accelerated Learning, in: Cohen, W.W., Hirsh, H. (Eds.), Machine Learning Proceedings 1994. Morgan Kaufmann, San Francisco (CA), pp. 181 – 189. URL: http://www.sciencedirect.com/science/article/pii/B9781558603356500301, doi:10.1016/B978-1-55860-335-6.50030-1.

[38] Miller, T., 2019. Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence 267, 1–38. Publisher: Elsevier.

[39] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., others, 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. Publisher: Nature Publishing Group.

[40] Moore, J.D., Swartout, W.R., 1988. Explanation in expert systemss: A survey. Technical Report. UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST.

[41] Murphy, K.P., 2013. Machine learning : a probabilistic perspective. MIT Press, Cambridge, Mass. [u.a.]. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

[42] Nicas, J., Kitroeff, N., Gelles, D., Glanz, J., 2019. Boeing Built Deadly Assumptions Into 737 Max, Blind to a Late Design Change. The New York Times URL: https://www.nytimes.com/2019/06/01/business/boeing-737-max-crash.html. [Accessed 2020-05-26].

[43] Oliphant, T., 2006. NumPy: A guide to NumPy. URL: http://www.numpy.org/. published: USA: Trelgol Publishing.

[44] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F.d., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-lib pdf.

[45] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825–2830.

[46] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M., 2017. Parameter space noise for exploration. arXiv preprint arXiv:1706.01905 .

[47] Quinlan, J.R., 1986. Induction of decision trees. Machine learning 1, 81–106. Publisher: Springer.

[48] Quinlan, J.R., others, 1992. Learning with continuous classes, in: 5th Australian joint conference on artificial intelligence, World Scientific. pp. 343–348.

[49] Rajamani, R., 2012. Lateral vehicle dynamics, in: Vehicle Dynamics and control. Springer, pp. 15–46.

[50] Rorvik, E.L.H., 2020. Automatisk dokking av et autonomt overflatefartøy. Master's thesis. NTNU.

[51] Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach. 3rd ed., Prentice Hall Press, USA.

[52] Samek, W., Wiegand, T., Müller, K.R., 2017. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. _eprint: 1708.08296.

[53] Shapley, L.S., 1953. A value for n-person games. Contributions to the Theory of Games 2, 307–317.

[54] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms.

[55] Solheim, S., 2020. NTNU cancels all lessons on campus. URL: https://dusken.no/artikkel/29239/ntnu-cancels-all-lessons-on-campus/. [Accessed 2020-06-02].

[56] Sundararajan, M., Taly, A., Yan, Q., 2017. Axiomatic Attribution for Deep Networks. CoRR abs/1703.01365. URL: http://arxiv.org/abs/1703.01365. _eprint: 1703.01365.

[57] Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

[58] Tulio Ribeiro, M., Singh, S., Guestrin, C., . LIME (Local Interpretable Model-Agnostic Explanations,). URL: https://github.com/marcotcr/lime.

[59] Tulio Ribeiro, M., Singh, S., Guestrin, C., 2016. “Why Should I Trust You?”: Explaining the Predictions of Any Classifier. arXiv e-prints , arXiv:1602.04938_eprint: 1602.04938.

[60] Udacity, I., . Deep Deterministic Policy Gradients (DDPG). URL: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum.

[61] Uhlenbeck, G.E., Ornstein, L.S., 1930. On the Theory of the Brownian Motion. Phys. Rev. 36, 823–841. URL: https://link.aps.org/doi/10.1103/PhysRev.36.823, doi:10.1103/PhysRev.36.823. publisher: American Physical Society.

[62] Van Lent, M., Fisher, W., Mancuso, M., 2004. An explainable artificial intelligence system for small-unit tactical behavior, in: Proceedings of the national conference on artificial intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. pp. 900–907.

[63] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, , Feng, Y., Moore, E.W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., Contributors, S..., 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17, 261–272. doi:https://doi.org/10.1038/s41592-019-0686-2.

[64] Wickramarachchi, D., Robertson, B., Reale, M., Price, C., Brown, J., 2016. HHCART: An oblique decision tree. Computational Statistics & Data Analysis 96, 12–23. URL: http://dx.doi.org/10.1016/j.csda.2015.11.006, doi:10.1016/j.csda.2015.11.006. publisher: Elsevier BV.

[65] Wiewiora, E., 2010. Reward Shaping, in: Sammut, C., Webb, G.I. (Eds.), Encyclopedia of Machine Learning. Springer US, Boston, MA, pp. 863–865. URL: https://doi.org/10.1007/978-0-387-30164-8_731, doi:10.1007/978-0-387-30164-8_731.

[66] Wong, A., . Model Tree. URL: https://github.com/ankonzoid/LearningX/tree/master/advanced_ML/model_tree.

[67] Yurtsever, E., Lambert, J., Carballo, A., Takeda, K., 2020. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. IEEE Access 8, 58443–58469.

[68] Zhang, S., Yao, L., Sun, A., Tay, Y., 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Comput. Surv. 52. URL: https://doi.org/10.1145/3285029, doi:10.1145/3285029. place: New York, NY, USA Publisher: Association for Computing Machinery.

Appendices

Appendix A

Additional figures of Linear Model Trees

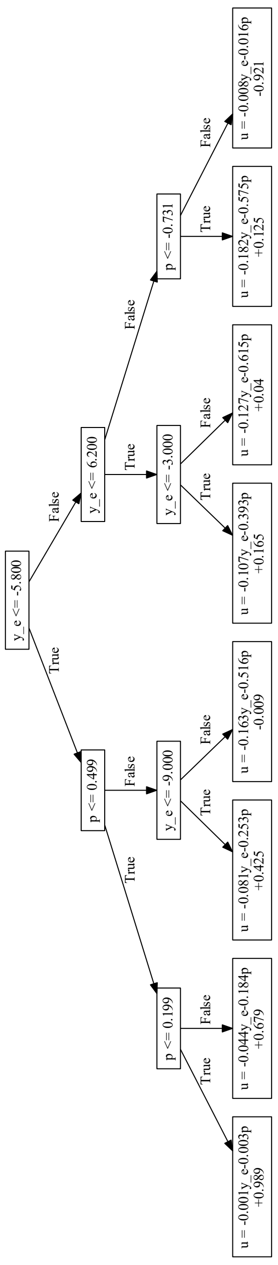

y_e <= -5.800
True
False
p <= 0.499
y_e <= 6.200
True
False
True
False
p <= 0.199
y_e <= -9.000
y_e <= -3.000
p <= -0.731
True
False
True
False
True
False
True
False
u = -0.001y_e-0.003p +0.989
u = -0.044y_e-0.184p +0.679
u = -0.081y_e-0.253p +0.425
u = -0.163y_e-0.516p -0.009
u = -0.107y_e-0.393p +0.165
u = -0.127y_e-0.615p +0.04
u = -0.182y_e-0.575p +0.125
u = -0.008y_e-0.016p -0.921

Figure A.1: The Linear Model Tree depth 3

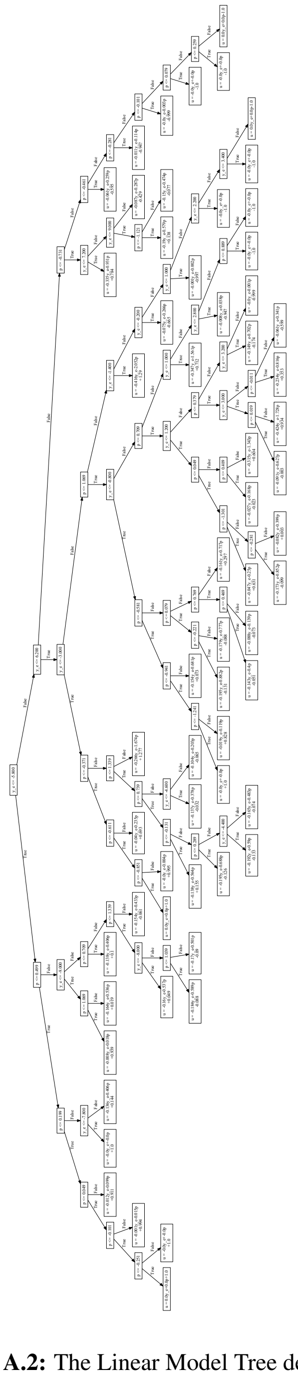


Figure A.2: The Linear Model Tree depth 10