Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

NEURAL TREE DISTILLATION TO EXPLAIN DEEP REINFORCEMENT LEARNING POLICIES

Senne Deproost

2020-2021

Supervisor: prof. dr. Ann Nowé

Advisor: Youri Coppens

Sciences and Bio-Engineering Sciences

Proef ingediend met het oog op het behalen van de graad van Master in de Ingenieurswetenschappen: Computerwetenschappen

NEURALE BOOM DISTILLATIE VOOR HET VERKLAREN VAN DEEP REINFORCEMENT LEARNING STRATEGIEËN

Senne Deproost

2020-2021

Promotor: prof. dr. Ann Nowé

Begeleider: Youri Coppens

Wetenschappen en Bio-ingenieurswetenschappen

Deze masterproef is (ten dele) tot stand gekomen in de periode dat het hoger onderwijs onderhevig was aan een lockdown en beschermende maatregelen ter voorkoming van de verspreiding van het COVID-19 virus. Het proces van opmaak, de verzameling van gegevens, de onderzoeksmethode en/of andere wetenschappelijke werkzaamheden die ermee gepaard gaan, zijn niet altijd op gebruikelijke wijze kunnen verlopen. De lezer dient met deze context rekening te houden bij het lezen van deze masterproef, en eventueel ook indien sommige conclusies zouden worden overgenomen.

This masters thesis came about (in part) during the period in which higher education was subjected to a lockdown and protective measures to prevent the spread of the COVID-19 virus. The process of formatting, data collection, the research method and/or other scientific work the thesis involved could therefore not always be carried out in the usual manner. The reader should bear this context in mind when reading this Master's thesis, and also in the event that some conclusions are taken on board.

Abstract

In recent years we saw the rise of self-adapting computer algorithms take over as a widely used form of automation. Most often these smart programs are heavily influenced by machine learning (ML), a domain within the field of artificial intelligence. The goal of ML is to train a learning algorithm on experiences to be able to perform a task. Within ML, several sub-domains are present. Algorithms can both learn with or without the supervision of an overseer. These are respectively called supervised and unsupervised learning. A third category, reinforcement learning (RL), uses the environment where it operates in to gain feedback in the learning process. The merger of RL together with the human brain-inspired deep learning approach gave rise to a method called deep reinforcement learning (DRL). Many applications, such as self-driving cars and robot control, are recently developed to take advantage of the DRL setup. However, just as the brain itself, the DRL algorithms are difficult to be interpreted by a human being. This so-called "black box" problem doesn't provide any meaningful insights in the model's inner workings causing difficulties in understanding, debugging and trusting the system.

In the field of explainable artificial intelligence (XAI), and by extension explainable reinforcement learning (XRL), techniques are developed to enhance the interpretability (and explainability) of a model. With a technique called knowledge distillation, the behaviour from one black box model could be transferred to a interpretable surrogate model, mimicking the behaviour of the black box model. Recently, this has been done with the distillation of DRL policies into a type of neural tree. The used tree model, called soft decision tree (SDT), was capable of mimicking the original DRL policy in a gaming environment, making the policy more interpretable due to its simpler structure.

A disadvantage of using an SDT is its static structure when initialised. A better approach would be to learn the optimal tree architecture during the training process. Adaptive neural tree (ANT) is a type of neural tree model capable of learning an optimal structure.

In this thesis, we wanted to make an initial incorporation of ANT into a XRL context by making several contributions. First, we developed a framework to facilitate the distillation process and for the visualisation of the learned structure and features. Secondly, we trained several SDTs and ANTs on games in the OpenAI Gym and we compared both models on both performance and interpretability criteria. The scores of the algorithms are benchmarks for performance while the visualisations of the tree, in combination with a newly developed complexity measurement, provides a measurement of interpretability.

For one of our evaluated environments, we can conclude that both ANTs and SDTs are capable of reproducing the DRL policy. For other gaming environments, the distillation couldn't provide a performant surrogate model. However we do analyse the learned behaviour and compared the models. Observable is a trade-off of reduced performance compared to the original DRL policy in exchange for increased interpretability of the tree models. In comparison to each other, the largest trained SDTs can outperform ANTs but this depends on the learned task or environment. When compared to the relative performance per node in the tree, we can state that ANTs perform better with the same number of nodes. For the interpretability criteria, most often the ANT is less complex in its structure.

However, the structure of the ANTs overdepends on the neural network classifiers in the leafs and therefore internal routing is less obvious. To resolve this, we introduced a technique called *smart routers, dumb solvers* (SRDS) as a third contribution to enforce the internal nodes to train more. This made the internal routing of the input frames more understandable in ANTs.

Acknowledgements

To start, I want to thank my supervisor prof. dr. Ann Nowé and advisor Youri Coppens for giving me the opportunity of doing this thesis. The past year and a half since I started writing were challenging. Throughout these difficult periods, the advise and help I received were indispensable in following the path towards researching, reasoning and eventually writing this thesis. I'm grateful for every minute spent (online) meeting and correcting the manuscript as well as the support for continuing until the end.

I sincerely want to thank my incredible mothers for the upbringing, love and support over the years. It was not easy during the past months. Yet, even in those times, they were always there for me. Even when I didn't know I needed a helping hand, I could always find it in the comfort of their presence.

I want to thank my friends for the support as well. It always cheered me up when we came together (in limited gatherings of course) and it always charged me with positive energy.

Finally, I want to thank every docent and teacher I had over the years from Koninklijk Instituut Woluwe, Sint-Jozef Ternat and Vrije Universiteit Brussels. They learned me to stand on my own feet in this world both as an individual person and as a member of society.

We are living in challenging times with even more difficult problems to solve. Let us therefore remember the adage this university is built upon and build further towards a bright future:

Scienta vincere tenebras.
Conquering darkness by science.

Senne Deproost

Acronyms

A2C Advantage Actor Critic.

A3C Asynchronous Advantage Actor Critic.

ACG Automatic Code Generation.

AI Artificial Intelligence.

ANN Artificial Neural Network.

ANT Adaptive Neural Tree.

DL Deep Learning.

DQN Deep Q Network.

DRL Deep Reinforcement Learning.

LMS Least Mean Squares.

MDP Markov Decision Process.

ML Machine Learning.

MLP Multi-layer Perceptron.

MSE Mean Squared Error.

NN Neural Network.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SDT Soft Decision Tree.

XAI Explainable Artificial Intelligence.

XRL Explainable Reinforcement Learning.

Contents

1 Introduction **5**
1.1 Machine Learning 6
1.2 Interpretability in machine learning 7
1.3 Problem statement 9

2 Reinforcement learning **10**
2.1 Fundamentals 11
2.1.1 The RL framework 11
2.1.2 Markov decision process 13
2.1.3 Value functions 14
2.1.4 Exploration-exploitation trade-off 15
2.1.5 Temporal difference 16
2.1.5.1 On-policy 18
2.1.5.2 Off-policy 19
2.1.6 Policy-based RL 19
2.1.7 Continuous input space 21
2.2 Deep Reinforcement Learning 22
2.2.1 Deep Learning 22
2.2.2 Algorithms for Deep Reinforcement Learning 24
2.2.2.1 Value-based DRL: Deep Q-learning 25
2.2.2.2 Actor-Critic methods: A3C 25
2.2.2.3 Policy-based DRL: Proximal Policy Optimization 26

3 Explainable AI **28**
3.1 Motivations 30
3.2 Use cases 31
3.3 Performance-readability trade-off 33
3.4 Taxonomy 34
3.5 Conventional techniques 43
3.5.1 Inherently interpretable model: decision trees 43
3.5.2 Post-hoc representation: feature visualisation 45
3.5.3 Additive explaining model: rule list 47
3.6 Knowledge distillation 48
3.6.1 Soft decision tree 50
3.6.1.1 Training a SDT 51
3.6.2 Adaptive Neural Tree 53
3.6.2.1 Training an ANT 55

3.7 Explainable Reinforcement Learning 57
3.7.1 Transparent algorithms 57
3.7.2 Post-hoc explainability 58
3.7.3 Other literature 58

4 Methods and setup **59**
4.1 Experimental setup 60
4.1.1 Training and policy selection 60
4.1.2 Knowledge distillation 60
4.1.3 Analysis 61
4.2 Network and policy architectures 62
4.2.1 The Deep Q-value network 62
4.2.2 The synchronous Actor-Critic model 63
4.2.3 PPO parameters 64
4.3 Models and adaptations 65
4.4 Measuring complexity 67
4.5 Prototype Framework 70
4.5.1 Technical details 70
4.5.2 Web interface 70

5 Evaluation **74**
5.1 Motivation and goal 75
5.2 Environment simulation 76
5.2.1 OpenAI Gym 76
5.2.2 Preprocessing for DRL 77
5.2.2.1 Preprocessing for Neural Trees 81
5.3 Experiments 82
5.3.1 MNIST dataset 82
5.3.2 Ms Pacman 93
5.3.3 Enduro 111
5.4 Results 117

6 Conclusions **118**
6.1 Contributions 119
6.2 Discussion 120
6.3 Future work 121

A Graybox **122**

B MNIST **124**

C OpenAI Gym **128**

Chapter 1

Introduction

Computers are wonderful machines. It is almost impossible to remove them out of our daily life without sacrificing a certain degree of comfort, convenience and luxe we all expect. They are powerful devices that allow us to program automation with code rather than build it from mechanical fine tuning, often with complex specialized hardware. Flexible tools with human readable interfaces to facilitate the coding of their inner mechanisms, giving it the power to do our bidding. Controlling vast networks of systems like the internet or insuring the precise dosage of medicine to a patient. Steering large dump trucks in open air coal mines or performing lightning fast exchanges on the stocks market. Their use cases are endless yet the need for their integration far outweighs the available human expertise to program them. To write efficient, performant and safe code that works as expected one should learn the art of software engineering or at least specialize in programming within a certain language or framework. However, it can not be expected from every programmer to deliver the same high quality of code for every case. The mythical 10x developer, capable of doing 10 times the work of an average developer with the same pristine quality is a very high standard to reach and is completely unrealistic. When quantity outweighs quality, bugs and errors are inevitable. Unintended behaviour of the software system could result in dangerous situations when dedicated to healthcare or other delicate domains. The emergence of Automatic Code Generation (ACG) could be an answer to facilitate software production [1]. With it, code is produced using several generative techniques and based upon a given software architecture the user requires from the system. However it has the disadvantage to create large amounts of repeatable code that are difficult to maintain by a programmer. Lacking clear separation in several files, it is more often uncertain which part is human written and what are machine generated lines of code. Furthermore, it is quite an investment to switch to a generative scheme for partly or full development of a system [2]. It is therefore impossible to write software dealing with every scenario a computer controlled device could encounter. If not counted for, the device will handle its tasks incorrect when it is outside the intended scope. That is why we see powerful factory robot arms assembling cars behind cages and big red emergency buttons next to them. When introduced to a unprogrammed situation like a human standing in the way it would just operate as normal, harming the person in a best case scenario. Components like safety rails and obstructions are needed to ensure demarcation from the operating area. Rather than handling this safe-to-be-sure approach, we could create adaptive systems that change their behaviour based on the situation currently ongoing. Remind that we cannot hardcode the system for every possible situation, so we have to rely on a scheme to find a suitable handling by the system. This could be accomplished by a form of *learning* that the machine can do in such a scenario.

The field of *Artificial Intelligence* (AI) tries to bring awareness, reasoning and adaptation to the computer. It is the study of understanding intelligence and how it could be built in a system, allowing it to be used in different specialized applications [3]. An individual instance of AI is called an intelligent *agent*. This agent uses a computational approach to reasoning in order to fulfil an assigned task. A task could be one we humans know the answer to, like how to assemble a car, or a task for which we don't know the answer, like finding where in the universe exoplanets could reside. Most of these unknown tasks are ones regarding search and optimization.

1.1 Machine Learning

Machine Learning (ML) is a subfield of AI where the agent tries to improve automatically by gathering experience [4]. This experience most often comes in the form of a *dataset* where a computational *model* can learn from. The goal is to create a so called *learning system* that can learn correlations in a given dataset. In the past, most of these correlations were handwoven into the system with logic rules. Nowadays, thanks to the advent of internet and smart mobile devices, we can use a huge amount of available data to automatically train the model. If we have a look at ML as a domain, we can distinguish several other subdomains as shown in figure 1.1.

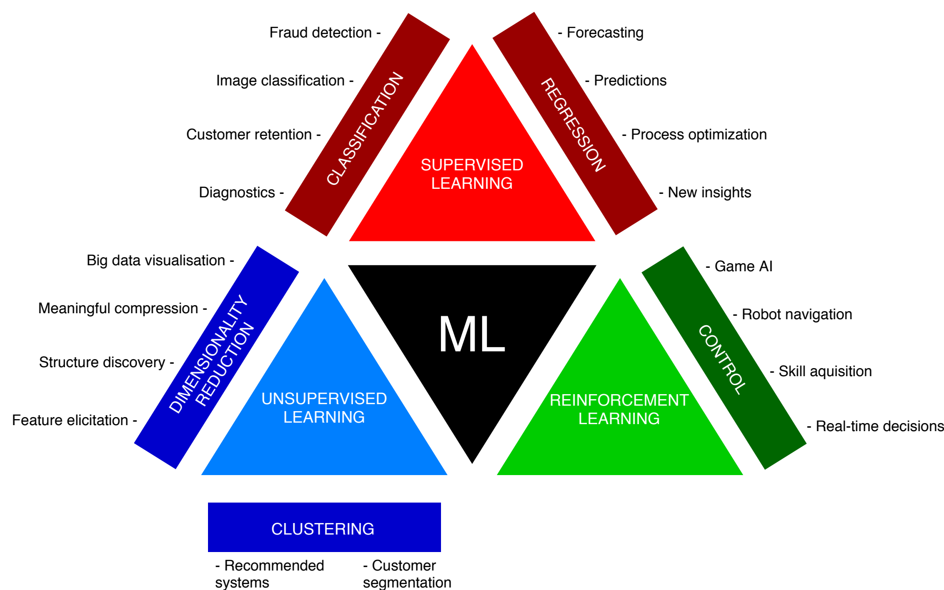


Figure 1.1: An overview of subdomains within the machine learning domain. [1]

In **supervised learning** the model receives feedback from a supervisor during learning. This is most often done with the use of labeled datasets where for each given input x a target y is specified to be the correct prediction. Two distinctive learning tasks within the subdomain are **classification** and **regression**. The first is a problem where the model has to predict one of

[1]Source: https://medium.com/ml-research-lab/machine-learning-algorithm-overview-5816a2e6303

several countable outcome classes for any given input it gets. A popular application is the recognition of objects in images and the detection of medical conditions from patient data. Regression is the prediction of a continuous function given input data. It is most often used in financial and meteorology forecasting and optimization problems. **Unsupervised learning** lacks supervision and can learn a task completely based on a non-targeted dataset. Here **clustering** tries to group inputs most similar to each other in relational clouds. From these clusters meaningful information about the input's distribution could be examined. Marketing and recommendation systems heavily rely on clustering for their predictions. **Dimensionally reduction** reduces the complexity of inputs to bring new insights in their relations. Most often big data visualization and compression tasks take advantage of this technique. At last, we have the subdomain of **reinforcement learning** (RL) and the main focus of this thesis. RL is neither supervised or unsupervised learning as the role of the supervisor is completely omitted. Instead the model relies on the feedback and experiences it makes in an environment. Game AI and robotics are well known applications of RL.

In the past decades, RL has proven to be a potential powerful technique to tackle on several ambitious problems. From self-driving cars [5] to robotic control [6] to defeating the best go player in the world [7]. These applications are challenging to learn in a supervised or unsupervised learning context because of the difficulty of generating enough and relevant data to be used as input. The environments where the algorithms operate in are often too complex to be generated by hand. Instead, models have to learn from observations in simulators or in the real world used to generate input data. The state-of-the-art RL techniques uses a deep neural network to learn their behaviour. This is a supervised learning technique inspired by the human brain. However, just like the brain, these models are difficult to be understood by a user. This makes adapting and controlling these so-called deep reinforcement learning (DRL) applications very challenging.

1.2 Interpretability in machine learning

The structures we use to learn a ML task can be varied. Each model's structural representation can be described by parameters. By examining these values one can retrieve meaningful insight on how the model handles the learning task. However not every structure has a representation that can be easily understood by a human. In fact, it is quite difficult to explain the agent's behaviour when dealing with models of high complexity. Take for instance the example of an artificial neural network. Its parameters decide both the structure composition (how many layers and neurons are included, the type of nodes, ...) as well as the individual behaviours of its components (weights, biases, activation function, ...). One could understand the behaviour of a single perceptron, as shown in figure 1.2 but would comprehend less of the multi-layer variant due to the increase in descriptive variables.

We could define *interpretability* as the degree to which a human can understand the cause of a decision by the model [8]. It also means someone could consistently predict results up to a certain degree, allowing the user to comprehend why specific decisions made by the model. *Explainability* implies that an explanation exists for the behaviour of the model. This is achieved when an answer to a why-question can be provided [9]: why does the model prefer one class above the other and why does it come up with a certain regression value? Both terms are closely related but differ in means of time and way information is provided. While interpretability is considered as a priori insight, meaning the inner workings of the system are observable given the input, explainability gives a posteriori understanding by using a second method to generate an explanation [10]. When a model's inner workings cannot be explained to the user or understood

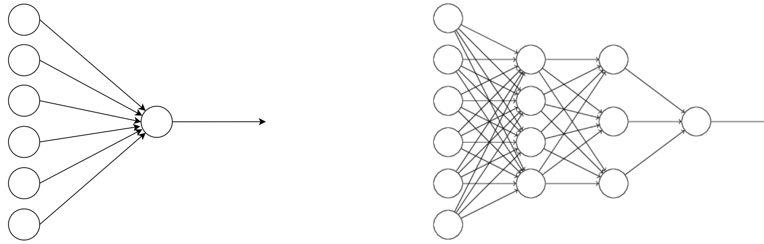


Figure 1.2: A single perceptron vs a multi-layer perceptron (MLP). Both models have 6 input nodes and one output. The one on the left needs 6 weights parameters while the one on the right needs 39, increasing its complexity.

by a domain expert it is a *black box* type model. Some examples include artificial neural networks and several ensemble methods. If the model can provide results associated to their behaviour that are at least comprehensible by an expert, then it is of the *white box* kind. These include decision trees and linear regression models. While a white box allows the exploitation of a prior knowledge it will perform weak in situations where randomness is present in the object and its environment. This is an advantage in black box cases, using statistical methods, but these have the disadvantage of generating reproducible results without full certainty [11]. Between these two there exists a *grey box* category representing systems that contain information about their behaviour but still need to be tested against experimental data [12]. In thesis we focus more on the explainability of black box models not in a white but grey kind of manner. We want to use an interpretable machine learning structure who's internal decision making isn't fully interpretable but gives meaningful insights into the decision making process.

Why interpretability could be important in a model is motivated by several reasons [13, 8]:

- One of the most important reasons for introducing interpretability is the detection of faults and subsequent debugging of the system. If its behaviour could be interpreted the resulting response can be easier examined against what is expected from the model.
- For many users there is still a lacking confidence in made predictions by a machine. This social acceptance could be improved by giving an open view inside the algorithm or by analogous explaining what the machine is doing/thinking in a simplified manner.
- Made decisions are subject to bias when the latter is present in the training dataset. This could lead to discriminating predictions against certain ethnicities or minorities. Necessary tweaking to the model should be done to counteract this bias in order to obtain fair results.
- When deployed in delicate situations like on public roads or inside a human body we want to guarantee a high degree of safety. Human assessment could be necessary in the system when we can detect and correct those parameters that lead to unsafe handling.
- It is also from a scientific point of view interesting to examine complex models like neural networks while also being able to point out the parts needed for certain actions. This stems mainly from human curiosity and the nature of understanding how we learn and why we behave as we do.

1.3 Problem statement

This thesis builds further on work done by Coppens et al. on the use of interpretable ML structures to learn DRL policies, specifically *decision trees* [14]. Their work focused on the use of *Soft Decision Trees* (SDT) [15] as a model for learning to play levels in the *MarioAI* environment [16]. This was accomplished by using a technique called *Knowledge Distillation* [17] to transfer a learned policy from a deep neural network to a SDT. Coppens et al. have shown that this creates usable models while achieving a degree of interpretability by visualizing learned weights of the tree nodes. These visualizations are represented as heatmaps of the nodes and can be combined with tile maps to bring meaningful insight in the agents behaviour. This increase in explainability comes at the cost of delivering slightly less performant models compared to the original trained neural networks. The authors remarked the shortcoming of SDTs of having a predefined depth which could result in a non-optimal tree structure. A proposed solution is the use of adaptive growing structures called Adaptive Neural Trees (ANT) [18]. Here the user can decide upon the balance between interpretability and complexity of the model by specifying how long the model could learn its optimal structure.

With this thesis, we tried to accomplish several objectives. The main goal was examining the usability of ANTs and their performance both in terms of learning and interpretability. Secondly we wanted to generalize the use of decision trees to other environments compatible with OpenAI Gym, a commonly used training environment for RL policies [19]. To facilitate the learning process, we created a *command line interface* (CLI) program, called *Graybox*, to train, distillate and test DRL models. It accomplishes the need of only 3 commands in the terminal to go through all stages of distilling a SDT or ANT policy. This approach makes training models on external computing servers much easier and allows for an easy interface to further build upon. We developed a visual component of Graybox in the form of a small web application that can run in a browser. This allows us to show the learned structures of the trees as well as the real-time interactions with the Gym environment. At last we tackled on an inherent issue with ANTs and their interpretability. Because they learn with relatively powerful leaf nodes compared to the internals, parameters of these internals are not as much optimized. This causes the routing behaviour of the tree to be less deterministic, decreasing the interpretability of each of these components. To solve this, we introduce a technique called SRDS (smart routers, dumb solvers) to encourage an ANT to focus more on its internals while optimizing parameters during training.

We start by having a short introduction to the field of (deep) reinforcement learning and deep learning. Afterwards we continue the literature section describing the current state of *Explainable Artificial Intelligence* (XAI) followed by the description of the used environments in the project. We continue with the methodology and setup of performed experiments and the evaluation of our results. Lastly we dedicate a section to discussion of the work done as well as possible expansions for future endeavours.

Chapter 2

Reinforcement learning

Reinforcement learning (RL) is a subfield of machine learning that studies a computational approach to understanding and implementing goal-directed learning and decision-making techniques in intelligent systems [20]. Instead of elaborating on a learning method to solve a defined problem, RL lays focus on the problem itself to be resolved. Every technique that is capable in doing this could be considered as a reinforcement leaning technique. Compared to the *supervised learning* paradigm of machine learning RL differs in where received feedback, used by the agent to learn a certain desired behaviour, originates from. In the supervised case an external overseer judges the agent upon its decisions, traditionally if the predicted classification is correct or how close the regression approaches the target. This intervention is most often done by labeling a target value to each input in accordance to the preferences of the overseer. This labeling procedure follows the logic of a certain task the agent has to learn. It is this logic that will dictate the behaviour the agent can learn from a resulting dataset the user created. An RL program, also called agent, retrieves feedback of the current residing environment, shaped in the form of positive or negative feedback. This feedback is also prone to delay as it could be only given when a terminal state has been reached. Another characteristic of reinforcement learning in comparison to supervised learning is the balancing between exploration and exploitation. The objective of an RL agent is to learn the behaviour that would yield the highest positive feedback when achieving its terminal state. This type of feedback indicates the agent performs certain behaviours that are in accordance to the ultimate goal it has to learn. Most often this takes on the form of a numeric value called *reward*. The more reward an agent gathers, the more positive the feedback and the better it tries to achieve its expected goal. To obtain this an agent could exploit the knowledge from past experiences that yielded high rewards. In addition it could explore new behaviour that yields more reward than from knowledge of past behaviour.

The first mentions of Reinforcement Learning can be traced back to the first half of the 20th century. In his book *Animal Intelligence: Experimental Studies*, psychologist Edward Thorndike tried to explain the emergence of learned behaviour in animals [21]. He noticed that an animal can learn how it has to interact in its environment in order to gain recompense by recalling this experience numerous times. Each pass through of the situation reinforces the perception of both positive, rewarding interactions and negative, punishable ones. Eventually the animal will learn a series of actions to perform that will lead to the most efficient way of achieving its goal in the environment or the most profitable one based on stimuli like food or pleasure. This training by trial and error is what Thorndike calls *The Law of Effect* by reinforcing experiences through repetition. This term coins both the selective (seeking out new behaviours and comparing their

impact) and associative (linking the instances with specific situations) aspect of learning by trial-and-error. The principle of *natural selection*, which in the context of evolution theory and evolutionary computing is applicable, only stands on the first aspect while supervised learning is considered associative alone. The strength of both combined form the essence of the Law of Effect, linking both a reinforced memory process that recalls lucrative behaviours as well as a search process towards new ones. Aside from the biological approach of animal learning, RL roots from the rise of optimization techniques in dynamic systems. These optimal control problems seek approaches to maximize or minimize the outcome of these systems in function of time. One solution was the introduction of dynamic programming [22] and the subsequent Bellman equations. These were value functions describing the optimal return a certain system could have, indicating an optimal way to achieve this. Later, optimal control problems were expanded into a discrete stochastic variant by Bellman called *Markov Decision Processes* (MDP's), laying the fundamentals of how RL problems could be formulated [20].

2.1 Fundamentals

In this section, we give an overview of crucial terms used within RL. We discuss the general framework together with the logic behind concepts such as *return* and *value*. At last we immerse ourselves into the well-know methods of performing RL.

2.1.1 The RL framework

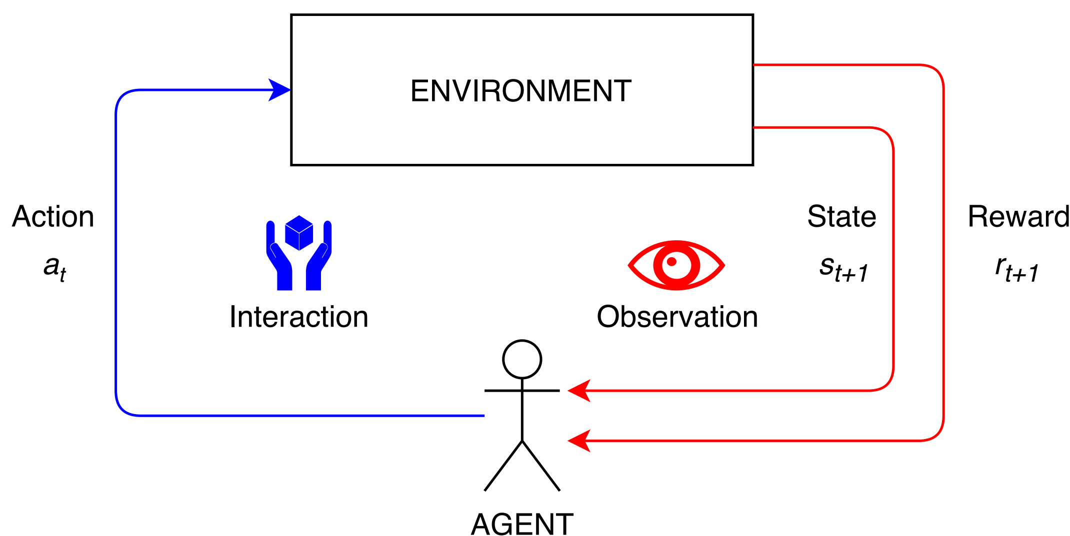


Figure 2.1: Schematic representation of an RL agent residing within an environment at time t. A step in an episode consists of an interaction with the environment followed by the observation of the new state and the accompanied reward. The time indicator t is incremented by one to $t + 1$ for taking a single step.

We describe the most basic elements of a reinforcement learning framework as described by [20]. We summarize the essential parts that make up the basic reinforcement learning model scheme as shown in figure 2.1. To provide a concrete example during this section, we will use the

case of a vacuum cleaner robot that has to navigate through the room with the goal of cleaning the entire room as fast as possible. Its actions will be the direction to go, indicated with going forward or backwards at a certain angle or to stop moving at all. Its start state will be the station where it can charge its batteries.

Environment
We model our intelligent instance as an software agent that has to learn and decide which behaviour to perform in order to achieve its goal. It therefore needs a time-bound space in which it could perform. We therefore say that the agent resides in an environment, implemented as either the real world or a simulation depending on the context of the problem. Current applications train the agent in a simulated environment and afterwards deploy it in the real world [23]. Training in a simulator has the benefit of being a magnitude faster compared to leaning in the actual world. In terms of descriptive variables, an environment could be either dynamic or static meaning the properties of the world could change or not respectively.

Episode
We define an episode $e \in \{1, 2, ..., E\}$ as part of a learning session. Every episode contains a series of steps indicated by a timestamp $t \in \{1, 2, ..., T\}$. The initial step is taken at the starting state s_0 of the agent and ends at s_T where the agent resides in a terminal state. The terminal state could be a goal state, indicating a successful episode, or a failure state, with possible punishment as a result. For a next episode in the session, t is initialized back to 0 and the agent in the starting state. For some RL applications, the use of episodes is discarded as it could be the learning of a continuing task where no clear end could be formulated. The final step T would be equal to ∞.

Action
Initially the agent has no model describing the environment and its details so it has to interact with it to gain information. This is done by performing an action $a_t \in A$, representing a chosen action at time t from the set of all possible actions A, and observing its effects in the environment. An action can be described at a high or low level of implementation. In the case of our vacuum robot this could be "navigate to the door" or "direct 5 volts of power to the left wheel's motor" respectively. This set is predefined before the agent starts learning but could be regulated with constraints and extensions depending on which RL model is followed. This is mostly done in real life applications where constraining is seen as a form of safety or when the agent has the ability to create higher level actions from a combination of many lower level actions.

State
A state is a signal from the the environment to the agent containing information about properties of that environment. An agent resides in a current state $s_t \in S$ (with S being the collection of all possible states) which can be discrete or continuous. Discrete observations are part of a countable set of possible states while continuous ones aren't. In the latter case, an observation is described in terms of continues variables like position and power. States could also be described at different levels of abstraction just like actions do. A state could be described as numerical values like sensory data or as situations like "cleaning carpet". When an interaction a_t has been made, the agent can observe two variables from the environment: a perception of the next state $s_{t+1} \in S$ and a reward $r_t \in R$ (with R the collection of all possible rewards).

Reward
The reward is an instance of the reward function, a mapping of the state to a number. This number represents the feedback of entering the new state s_{t+1} and is a measurement of intrinsic desirability of that state [20]. Reward indicates which actions are good or bad in a certain state. It is the objective of the agent to maximize its cumulative reward during an episode. Formulating a reward function, that is deciding when reward is given or not to the agent in a state, is non-trivial and could be approached by several strategies. The reward function can be implemented with sparsely signals (minimal feedback or only at a terminal state) or generously (in almost every state). Punishment in the form of negative reward can also be used when achieving a terminal state that isn't a goal state. When time-efficiency should be considered, a punishment per step can be accumulated to stimulate the agent of finding the best policy with as little steps as possible.

Policy
The behaviour of an agent can be formulated as the action it most likely will take in every individual state. This is also known as the policy $\pi_t(s, a)$, a function that maps the chance of taking action a in state s at time step t. Intuitively, at state s_t, the action a_t will be the one with the highest chance in accordance with the policy π_t.

Return
As stated before, the objective of the agent is to maximize its rewards while trying to reach a goal state. Preference goes to gaining the highest accumulative reward over the run of a session rather then immediate one on short term. The expected return is the sum of rewards $R_t = r_{t+1} + r_{t+2} + ... + r_T$, indicating the future cumulative sum from timestamp t till T. A better indication of return is discounted return

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

where γ represents the discount factor with $0 \leq \gamma \leq 1$. This formulates return k steps in the future as the sum of all expected rewards, weighted by a factor of γ. The further we count a reward, the higher k is and the higher the discount will effect that value. Choosing γ has an impact on what kind of reward the agent will focus. If $\gamma = 0$, the agent is "short-sighted" and is only concerned with immediate reward. All other values will ensure that the agent considers the future as well. Discounted return guarantees that even when dealing with an infinite sum, the resulting value for the sequence will always be finite.

2.1.2 Markov decision process

A state signal should contain information from immediate senses, like sensory data, and more. A state could include details about past signals and therefore a sort memory of past interactions. Differently, not everything could be informed by the state, even when it is favorable in the decision making process of the agent. If our robot cleaned half the area and resides in a corner, the state signal could reveal its position in the room. What the signal couldn't provide is the presence of walls that form the corner (if it has no proximity sensor for example). In this case the agent can only learn that at this position its direction should be changed because it wouldn't advance by going forward. If walls are detected it could learn to change direction based on this alone. Another example, where the robot is wall-sensitive, it could detect a cul-de-sac. The state still only contains position and wall placement, but implicitly it also contains information

of being a dead end. The best action to take is to turnaround and go into opposite direction to escape this dead end. We don't expect to know the sequence of every previous state we visited like a path since this would require more memory and brings computational overhead traversing the sequence every time the agent observes a state signal. In general, we expect from the state to be a summary of the past so it could learn and decide upon it in the future. A state capable of these expectations is called to have the *Markov Property* or is *Markovian.* This is a state that occurs independently of previous encountered states in the past. We can formulate an episode transition

$$Prob\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\right\} \tag{2.2}$$

as the probability of observing state s' with reward r at time $t+1$ when the history s_t, a_t, r_1, s_0, a_0 happened. If history doesn't effect the episode transition, we can formulate

$$Prob\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\right\} \tag{2.3}$$

indicating the transition is only dependent on the current state s_t when action a_t is performed. The state signal is Markovian if (2.3) and (2.2) are equivalent for all $s', r, s_t, ..., r_1, s_0, a_0$. This would allow us to predict the next s' and r as well as all other future states and expected reward by iterating on (2.3). This implies that choosing a best action in a state is the same as choosing it as the best action after a certain history. Defining the optimal policy becomes a search process that only has to consider the present situation, greatly simplifying the calculation of it.

When a reinforcement learning problem satisfies the Markov property, it is called a *Markov Decision Process* (MDP). Depending on having a finite state-action space or not, we can distinct between a finite and infinite MDP. The state transitions of such task can be modeled as a Markov Chain, summarizing all possible state transitions for the system as a directed graph. The probability of a transition occurring is equal to

$$Prob\left\{s_{t+1}=s' \mid s_t=s, a_t=a\right\} = \mathcal{P}^a_{ss'} \tag{2.4}$$

(the transition function of the MDP) and the the expected reward function

$$E\left\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\right\} = \mathcal{R}^a_{ss'} \tag{2.5}$$

MDPs containing states whose information is not fully observable are called *partially observable markov decision processes* (POMDPs). An example is a self-driving car on a road through a densely forest. An animal could suddenly appear from the bushes, trying to run across the asphalt. This is a signal the driver cannot anticipate on until the beast reveals itself.

2.1.3 Value functions

For indicating "how good" an agent is when in a certain state we can formulate a *value function* V^π. This calculation depends on expected return estimating how much future reward to gain starting from the current state s and following a policy π:

$$V^\pi(s) = E_\pi\left\{R_t \mid s_t = s\right\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{2.6}$$

This indicates a measure of future performance of the policy with E_π representing the expected value of that policy. When including an action variable a, we can define the Q-value function as

$$Q^\pi(s,a) = E_\pi\left\{R_t \mid s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \tag{2.7}$$

formulating an estimate more specific to the chosen action in the state.

Both value functions could be estimated purely from experience. For instance, we could calculate the return of a particular state by averaging all possible future rewards. The state's value function $V^\pi(s)$ when the number of passes of that state goes to infinity. When dividing averages based upon the action taken, we can approach $Q^\pi(s,a)$ as well. These methods of averaging random samples are called *Monte Carlo Methods* and are distinct from *dynamic programming* methods in that they don't require knowledge about the environment as a whole.

Value functions satisfy recursive properties that allow to derive the *Bellman equation* for V^π:

$$\begin{aligned}
V^\pi(s) &= E_\pi\left\{R_t \mid s_t = s\right\} \\
&= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_t = s\right\} \\
&= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right] \\
&= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]
\end{aligned} \tag{2.8}$$

The value function V^π is the only solution to its Bellman equation. Current state s values and the value of successor states s' are related in that the current state s_t can be valued by averaging the possible states S weighted by the probability of that successor state s_{t+1} occurring. It also concludes that the value of a current state is equal to the discounted value $\gamma V^\pi(s')$ of the expected successor state plus the reward $\mathcal{R}^a_{ss'}$ encountered.

When maximizing the expected return of an agent, we want to find a policy able to achieve this. A policy π is considered better when its return is higher than an other policy π' for every state. An *optimal policy* π^* is one that cannot be surpassed by any other policy. It is possible to have multiple optimal policies, only when their value function is the same. These policies are as well denoted by π^* with their optimal value function

$$V^*(s) = \max_\pi V^\pi(s) \qquad \text{for } s \in S \tag{2.9}$$

The optimal state-action value function is also shared:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \qquad \text{for } s \in S \text{ and } a \in A(s) \tag{2.10}$$

2.1.4 Exploration-exploitation trade-off

We denote the best action to take at a step t as a^*_t (2.11), which is the action that maximizes the Q-value at that step.

$$a_t^* = \arg\max_a Q_t(a) \tag{2.11}$$

Recall that the higher the Q-value of a state-action pair the more indication there is of being rewarded with high return in the future. It is straightforward that the agent has to *exploit* action a_t^* at any given moment. However, especially at the beginning of training, the agent doesn't have enough information of the environment to consider an action as the best one possible. Therefore it has to *explore* unseen actions at random to find possible better actions. This *exploration-exploitation* trade-off is determined by a *action selection strategy* [24]. The agent cannot rely on a action selection strategy purely based on either one of the two. If it only exploits best found actions, it will never find better possible actions not present in the policy. If it only explored then it doesn't care about best found actions, which are needed to increase accumulated reward.

A commonly used strategy is ϵ-greedy (2.12). In it the variable ϵ, with $0 \leqslant \epsilon \leqslant 1$, determines the probability of an action a_t at time step t being a random one to explore the action space. In all other cases, with a probability of $1 - \epsilon$, the best action will be exploited. To change the rate of exploration over time, we use a decay function that changes the setting from a exploitative policy at the start to a explorative one at the end. This could be a linear function that decreases the variable in function of t.

$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{2.12}$$

ϵ-greedy has the disadvantage of valuing all actions equally during an exploration move, even the non-optimal ones. A better strategy would be to select a random action with a probability based on the current estimate of that action. We could accomplish this by ranking the value-function estimates using a Boltzmann distribution [25]:

$$\pi(a|s) = Prob\left\{a_t = a | s_t = s\right\} = \frac{e^{Q_t(a)/\tau}}{\sum_{a'} e^{Q_t(a')/\tau}} \tag{2.13}$$

with τ as a positive parameter called temperature. A high temperature treats the actions more equal while low temperature has a more greedy action selection as result. τ can also incorporate a decay factor like ϵ-greedy.

2.1.5 Temporal difference

When trying to solve the RL problem we need a method to calculate the value function V^π (or Q^π when using state-action pairs). Two methods in estimating the value based on gathered experiences are *Monte Carlo methods* (MC) and *dynamic programming* (DP). Both techniques also solve the problem of having a delayed reward within the RL episode.

Dynamic programming
When a complete model of the environment is available to the agent dynamic programming can be used to form an optimal policy [26]. However, obtaining a perfect model (especially in the first episodes) is non-trivial and is resource demanding. From a theoretical standpoint DP provides the foundations of other value estimation methods.

Monte Carlo methods
When an incomplete view is available, MC can be used using nothing but experiences from averaging sample returns. No prior knowledge is necessary. MC relies on randomness to pick out the counted experiences [27].

TD
Reinforcement learning combines the concept of learning from random experiences using *Monte Carlo methods*(MC) with the sub-solution approach of *Dynamic Programming* (DP). The unification of both these techniques in RL is called *Temporal Difference* (TD) learning. Similar to Monte Carlo methods, TD learns from made experiences without having an internal model of the environment. Like dynamic programming, TD bootstraps from learned estimates to learn other estimates without arriving at a final state.

The estimation of the value function V^π at a given non-terminal state s_t is based on the gathered return after visiting that state. Monte Carlo methods only update the value when the return of the next step s_{t+1} is known, which implies that they can only estimate this when a terminal state is reached. If we formulate the update we get

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right] \tag{2.14}$$

with R_t as the return at t and α a step-size parameter for regulating the significance of each value update. This method of updating is called *constant-α MC*. On the other hand, TD methods are able to calculate the new value as soon as information of the next step is available.

TD(0) is the most simple method when given reward r_{t+1} in the next step

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{2.15}$$

with discount γ indicating the importance of the previous value in that state to calculate the new one. This *bootstrapping* method is similar to dynamic programming in that an existing local estimate is used to gain a global result. Where MC uses an estimation of (2.6), DP uses $V^\pi(s) = E_\pi \left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\right\}$. DP's target is an estimation because at current time $V^\pi(s_{t+1})$ is not known so $V_t(s_{t+1})$ is used. For MC the target is based upon the sample return at t. The TD target combines both methods forming an estimate V_t based upon the expected value (equation 2.6).

When using the TD(0) update rule in an algorithmic estimation of V^π we can write the following pseudo code:

Algorithm 1 Tabular TD(0) for estimating v_π [20]

Require: the policy π to be evaluated
1: Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
2: **repeat**
3:     **for each** episode **do**
4:         Initialize S
5:         **repeat**
6:             **for each** step of episode **do**
7:                 $A \leftarrow$ action given by π for S
8:                 Take action A; observe reward, R, and next state, S'
9:                 $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
10:                 $S \leftarrow S'$
11:             **end for**
12:         **until** S is terminal
13:     **end for**
14: **until** end of session

MC and TD are referred as *sample backups* because of the use of a succeeding state-action pair as a sample together with the sample's value and reward to create a backup value before changing the value of the pair. This sampled way of making backups differs from the DP approach where a complete probability distribution is used of all the possible succeeding states.

TD(0) has been proven to converge and will therefore always find an estimate for the MDP [28]. Other TD methods like *Sarsa* and *Q-learning* have been proven in the past to converge as well [29] [30].

2.1.5.1 On-policy

On-policy TD control uses the same policy to make decisions as the one it wants to improve. An algorithm for on-policy is *Sarsa* which uses the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ that makes up the transition from one state-action to another.

Algorithm 2 Sarsa: an on-policy TD control algorithm [20]

1: Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
2: **repeat**
3:     **for each** episode **do**
4:         Initialize S
5:         Choose A from S using policy derived from Q (e.g., ϵ-greedy)
6:         **repeat**
7:             **for each** step of episode **do**
8:                 Take action A, observe R, S'
9:                 Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
10:                 $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
11:                 $S \leftarrow S'$; $A \leftarrow A'$;
12:             **end for**
13:         **until** S is terminal
14:     **end for**
15: **until** end of session

Action-state values at time t are updated according to the following rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{2.16}$$

The part that makes this formula on-policy is the usage of $Q(S_{t+1}, A_{t+1})$ for the difference calculation within the update. This makes the algorithm use the available Q values of the current policy the agent possesses.

2.1.5.2 Off-policy

Off-policy TD control uses a different policy from the one that generates the experience [31]. The learned Q-value function can directly approximate the optimal value Q^*, independent of the used policy. The policy still has to decide which state-action pairs are visited and updated.

Algorithm 3 Q-learning: an off-policy TD control algorithm [20]

```
1: Initialize Q(s, a), ∀s ∈ S, a ∈ A(s), arbitrarily
2: repeat
3:     for each episode do
4:         Initialize S
5:         repeat
6:             for each step of episode do
7:                 Choose A from S using policy derived from Q (e.g., ϵ-greedy)
8:                 Take action A, observe R, S′
9:                 Q(S, A) ← Q(S, A) + α [R + γ max_a Q(S′, a) − Q(S, A)]
10:                S ← S′
11:            end for
12:        until S is terminal
13:    end for
14: until end of session
```

The update rule becomes:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{2.17}$$

Here the agent uses $\max_a Q(S_{t+1}, a)$ for the difference calculation. This makes the agent search for the action that maximizes the Q value of the next state S_{t+1}, making it off-policy.

2.1.6 Policy-based RL

TD-methods use value approximation to learn a state-value function. However, when dealing with a large or continuous action space, a value-based approach becomes less suitable because of the many values that need to be saved. It is also deterministic in providing only one optimal action as an outcome of the policy or several if similar optimal actions exist. Since the goal is to optimize the policy, an action-value estimation-based approach could be considered instead of following state-value estimation-based approaches [32].

Policy gradient-based algorithms are well known in the paradigm of policy-based approaches. In these a parametric model is used for the explicit representation of the policy. These parameters can be optimize with the goal of finding a policy with (sub-)optimal performance. They use gradient-ascent to optimize the policy approximation function and maximize the cumulative reward.

To measure the performance of a policy, we could define a policy value $J_{(\theta)}$ as follow:

$$J_{(\theta)} = \mathbb{E} \sum_{t \geq 0} \left[\gamma^t r_t | \pi_\theta \right] \tag{2.18}$$

θ is the policy parameter vector, containing weights, we want to optimize. To find the optimal parameter vector θ^* we have to satisfy $\theta^* = \arg\max_\theta J_{(\theta)}$. We define $\tau = \{(s_1, a_1, r_2), ..., (s_t, a_t, r_t)\}$ as the trajectory, which is the sequence of states visited in an episode. If we incorporate τ into (2.18) we could write the policy value as:

$$\begin{aligned} J_{(\theta)} &= \mathbb{E}_\tau \sum \mathbb{P}(\tau \nRightarrow \theta) r_{(\tau)} \\ &= \int_\tau r_{(t)} \mathbb{P}(\tau \nRightarrow \theta) \tau \end{aligned} \tag{2.19}$$

with $\mathbb{E}$ the expectancy of reward in a trajectory τ and $\mathbb{P}$ the probability of encountering τ under policy π_θ. In order to obtain the gradient of $J_{(\theta)}$ the equation (2.19) needs to be differentiated with respect to parameters θ:

$$\Delta_\theta J_{(\theta)} = \int_\tau r_{(t)} \Delta_\theta \mathbb{P}(\tau \nRightarrow \theta) \tau \tag{2.20}$$

The problem with (2.20) is the intractable nature of it which means there exists no mathematical formulation to solve this is an efficient manner. This is because we try to differentiate the function $p(\tau; \theta)$ over the vector θ whereas that function is conditioned on the same parameter [32]. If we simplify (2.20) in order to incorporate a Monte Carlo sampling part into it with expectancy $\mathbb{E}$, we get:

$$\begin{aligned} \Delta_\theta \mathbb{P}(\tau \nRightarrow \theta) \tau &= \mathbb{P}(\tau \nRightarrow \theta) \frac{\Delta_\theta \mathbb{P}(\tau \nRightarrow \theta)}{\mathbb{P}(\tau \nRightarrow \theta)} \\ &= \mathbb{P}(\tau \nRightarrow \theta) \Delta_\theta \log \mathbb{P}(\tau \nRightarrow \theta) \\ \text{therefore:} & \\ \Delta_\theta J_{(\theta)} &= \int_\tau (r_{(t)} \Delta_\theta \log \mathbb{P}(\tau \nRightarrow \theta)) \mathbb{P}(\tau \nRightarrow \theta) d\tau \\ &= \mathbb{E}_{\tau \sim \mathbb{P}(\tau \nRightarrow \theta)} \left[r_\tau \Delta_\theta \log \mathbb{P}(\tau \nRightarrow \theta) \right] \\ &\approx \sum_{t \geq 0} r_{(\tau)} \Delta_\theta \log \pi_\theta (a_t | s_t) \end{aligned} \tag{2.21}$$

An algorithm using this combination of policy-gradient and MC is the REINFORCE algorithm [33]:

Algorithm 4 REINFORCE [33]

1: Initialise θ arbitrarily
2: **for each** episode $\{(s_1, a_1, r_2), ..., (s_{T-1}, a_{T-1}, r_T)\} \sim \pi_\theta$ **do**
3:     **for each** $t = 1$ to $T - 1$ **do**
4:         $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta (s_t, a_t) v_t$
5:     **end for**
6: **end for**
7: Return θ

α is the step size of the gradient that could decay over time. Each step in the episode a gradient step is taken in the direction that maximizes the cumulative reward we would receive from the policy π_θ. For calculating the gradient a Monte Carlo sampling could be used. Here v_t represents a unbiased estimate sample of $Q_{\pi\theta}(s_t, a_t)$.

A shortcoming of REINFORCE is the high variance present in the algorithm. This could be attributed to two factors. One is the use of absolute rewards, resulting in varying results when using Monte Carlo to sample. The second is the attribution of reward to the specific state-action pairs that form a episode-trajectory. Rewards are averaged over all instances of the trajectory τ so it could be the impact of only a few good state-action pairs over a majority of lesser ones that causes high variance in the end. A method to reduce the variance is the use of a *baseline* which incorporates a discounted cumulative future reward function into the algorithm. One example of baseline is to take the advantage of a given action in a particular state, which is equal to the difference between the Q value $Q_{\pi_\theta,(s_t,a_t)}$ of that state-action pair with the value $V_{\pi_\theta,(s_t)}$:

$$\Delta_\theta J_{(\theta)} \approx \sum_{t\geq 0}(Q_{\pi_\theta,(s_t,a_t)} - V_{\pi_\theta,(s_t)})\Delta_\theta \log \pi_\theta(a_t|s_t) \tag{2.22}$$

2.1.7 Continuous input space

To keep track of the used approximation (value) function most often a tabular approach is considered to store the values. In the case of TD learning, a Q-table could be implemented as a simple two-dimensional matrix or dictionary depending on the used programming language. This approach is only applicable in discrete state- and action-spaces which is not always usable for real-world applications. Especially the input state-space could be large when dealing with image data from a camera sensor or sounds from a microphone. As shown in the policy-based RL method REINFORCE, we could use optimized parameters in a function to approximate the policy or value function. This can be both done in a linear and non-linear way. In the linear approach we could discretize the observation space using *tile coding* by dividing the space into a number of disjoint sets [34]. This method however is not injective as a tile encoding function ϕ does not imply $\phi(s) = \phi(s') \nRightarrow s = s'$ meaning this causes the MDP to become POMDP. A POMDP is a MDP which is only partial observable instead of being fully observable. Because of this, the agent has to make *belief* assumptions on these non-observable states that still could have an impact on the MDP process. One could also use fuzzy sets which introduce a fuzzy membership meaning that an element can belong partially in a set instead of being confined to false or true [35].

The disadvantage of using linear approximations is the need for good informative features, which may require hand-picking and domain knowledge [36]. The non-linear approach could approximate a function with better accuracy than a linear approximator using the same input features [37]. One example of non-linear approximator is the artificial neural network, which forms the basis for *deep reinforcement learning* (DRL).

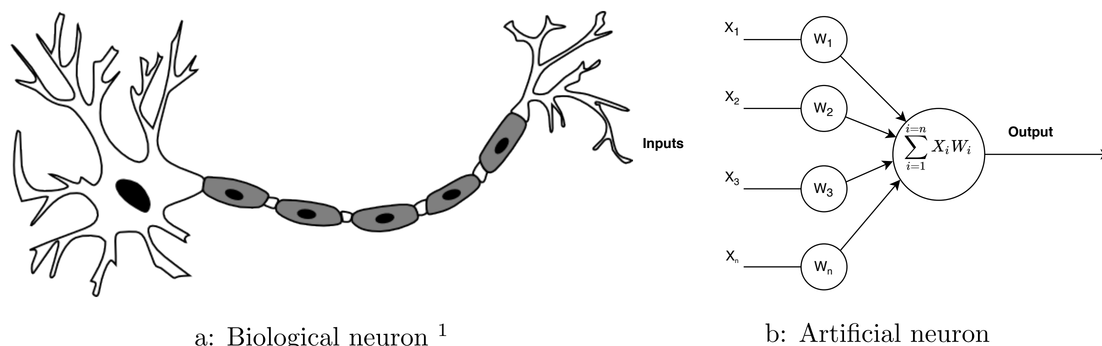
2.2 Deep Reinforcement Learning

Since its popularisation in 2015 by DeepMind [38], the incorporation of deep artificial neural networks into RL lead to the emergence of the field of *Deep Reinforcement Learning* (DRL). In the years after many achievements have been made using DRL ranging from defeating experts in games [7, 39] to vehicle and robot control [5, 6, 40] to infrastructure managing [41].

We introduce the basic concepts behind deep learning and discuss some well known DRL algorithms.

2.2.1 Deep Learning

We humans have always found answers and inspiration from this world we live in. To quote the Catalan architect Antonio Gaudi: "Nothing is invented, for it is written in nature first.". The same is true within the computer science and engineering community for domains like *Swarm Intelligence* [42], originated from the study of ant colonies, and *Robotics* [43], most often inspired by the study of the human body and animal movement. *Deep Learning* (DL) is a field of machine learning where computational models are based on the structure of the human brain [44]. With DL we try to approach the fundamental parts and interactions inside our head with a computational model which in turn can be fine-tuned to learn a certain behaviour.



a: Biological neuron [1]   b: Artificial neuron

Figure 2.2: Representation of a biological neuron from a human brain compared to a mathematical schematic of an artificial one.

Our brain consists of billions of specialized cells called neurons. These interact with each other in such complex manner intelligence can emerges from [45]. A signal can be sent from one neuron to another by means of connections called synapses. The neuron can receive signals from several neighbouring cells and could output a signal of its own when a threshold activation is reached. It is this kind of interactions DL tries to reproduce inside a computer.

The most simple structure, mimicking a biological neuron, is the perceptron [46] (figure 2.1b). This structure takes in several inputs X_i weighted by a factor W_i. A summation is made of all incoming signals, resulting in a single output by the neuron. The weights are variables we have to finetune, representing the importance of other connected neurons when used in a multi-layer context. To mimic the rate of action firing in a cell an *activation function* can be applied to the sum of signals [47]. In the case of a biological neuron this is the Heaviside step function [48] while a popular function for artificial neural networks (ANNs) is the Rectified Linear Unit (ReLU) and sigmoid function [49].

[1] Source: https://library.kissclipart.com/20180911/giq/kissclipart-motor-neuron-diagram-unlabeled-clipart-neuron-wiri-95afea43233b17ec.png

Groups of neurons can be divided into *layers* where each neuron has a one-way connection to neurons of a subsequent layer. Because of the unidirectional nature of the connections we can define an input and output layer in the case of a multi-layer perceptron (MLP) which is equivalent to biological senses (eyes, ears, nose ...) and actuators (like muscles and glands) respectively. When the number of layers is bigger than 2, the network contains *hidden layers*. This kind of architecture is also referred to as a deep neural network (DNN).

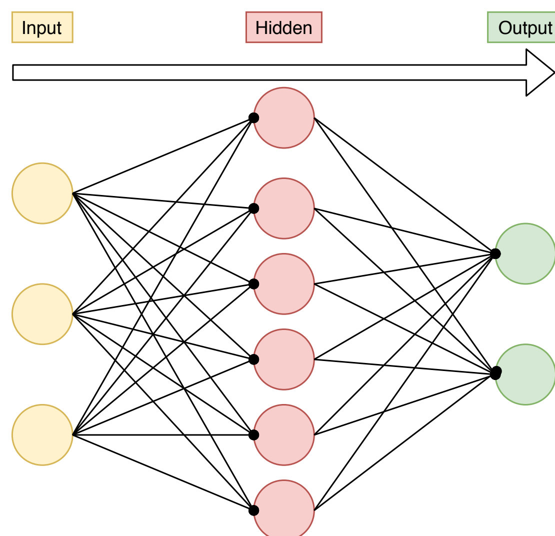


Figure 2.3: A simple 3 layer neural network consisting of an input, hidden and output layer. The arrow indicated the unidirectional flow of signals through the layers.

There exist different types of neural networks based on their layer compositions. If the layers only contain ordinary perceptrons then the network is classified as a feed-forward neural network. Convolutional neural networks (CNN) are a type of ANN that uses filters in the first layers of the architecture to mimic cortical neurons in the visual cortex of our brain [50]. These kind of networks are most useful in learning tasks where local features within the input data can be separated like in the case of image classification. A third kind of network is the recurrent neural network (RNN) based on neurons with an internal state [51]. Because these can memorize they are useful in cases where the input data is variable like speech and handwriting recognition.

As stated before, one of the variables in the networks are the weights of each connection. To find the most fitting values for these parameters, minimizing the loss function of the model given a certain learning task, we use a learning algorithm or *optimizer* [44]. *Gradient descent* [52] can be used to optimize by calculating the gradient of the weights but this is tedious in complex networks. Another method called *backpropagation* solves this by calculating the error at the end of the network while backtracking through it to update the variables [53]. Several other optimizers could be applied like Newton methods [54] and genetic algorithms [55]. Apart from regular nodes a layer could contain a node to regulate the bias in the model [56]. Bias can delay the triggering of the activation of the neurons, shifting the predictions and allowing a better fit on the given dataset.

Large networks can be powerful but come with the risk of overfitting the dataset [57]. While the training loss in a learning session can go down, the loss on never seen before data (validation or test data) could rise indication a decrease in generalization. The probabilistic removal of nodes

from a layer, called dilution or *dropout*, prevents the overdependence of the layer on several of its inputs [58]. By randomly masking nodes during training the network will consider all of the given inputs. We could also stop the training process when the validation loss starts to rise compared to training loss [59]. A threshold can be chosen indicating how many episodes the loss can go up or how much increase is tolerable. When allowed for, *data augmentation* or increase in the amount of data allows for a greater variance in the input. Data can also be generated by flipping or rotating images in the case for a image dataset.

Because of their use in DRL, ANN's could suffer from overfitting while learning a policy. However, the usual setup of RL problems consists of continuous learning tasks without a clear separation of training and test stages to detect this [60]. It is rather difficult to obtain generalization within a RL context because of the specific environment the agent is trained in. Changes like different dynamics, visual changes and structure variation can be present in the deployment environment leading to possible overreliance on the learned characteristics of the training environment [61]. We could inject stochasticity into the training environment with *domain randomization* [62] or by *frame skipping* [63]. As of today, no standardized way of performance measure or experimental protocol is used in the field.

2.2.2 Algorithms for Deep Reinforcement Learning

Like the more traditional form of RL, DRL has two main approaches to optimize a policy: with value estimation and via policy gradient. In between those two categories exists a third group of actor-critic methods who combine both the policy-gradient-based techniques with value-estimation [32].

For each of these categories we give a state-of-the-art algorithm example together with its main concepts. We note that for each type of algorithm many variants exists. This is because the DRL on itself is a recent field where many problems need to be yet solved. Different algorithms are developed to tackle on more complex environments and to reduce the computational resources needed to learn a policy. The problem of sample efficiency, where we want to only use a fraction of the data needed compared to the thousands of frames now, is also tackled on in several of these newly developed algorithms.

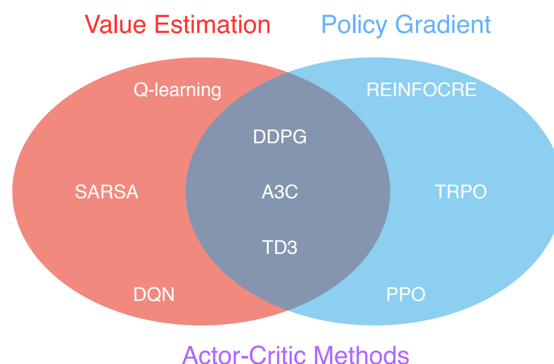


Figure 2.4: Overview of the three main approaches to (D)RL. Several algorithms per category are represented as examples.

2.2.2.1 Value-based DRL: Deep Q-learning

As stated before, the popularisation of DRL came with the introduction of DeepMind's *Deep Q-learning*, also called *Deep Q-Network* (DQN) [64, 38]. Instead of using a tabular approach, the algorithm uses a neural network as a value approximator to store the Q-values. This network could be an FNN or a CNN with the latter used in the original paper to learn playing video games on a Atari 2600 console.

Made transitions are stored in memory $\mathcal{D}$ during the episode. These *experiences* are used to train the DQN in batches, a commonly used tactic for updating ANNs [65]. The action selection is done by a ϵ-greedy policy. The gradient-descent minimizing the loss in the network is formulated as:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\varepsilon} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \tag{2.23}$$

Algorithm 5 Deep Q-learning with Experience Replay [64]

1: Initialize replay memory $\mathcal{D}$ to capacity N
2: Initialize action-value function Q with random weights
3: **for each** episode $= 1, M$ **do**
4: Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
5: **for each** $t = 1, T$ **do**
6: With probability ϵ select a random action a_t
7: otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
8: Execute action a_t in environment and observe reward r_t and image x_{t+1}
9: Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
11: Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
12: Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
13: Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to (2.23)
14: **end for**
15: **end for**

2.2.2.2 Actor-Critic methods: A3C

The Actor-Critic architecture, as the name implies, is comprised of two main parts [32]. The *actor* is a model suggesting actions the agent could take according to a certain policy. Meanwhile the *critic* will give feedback on the chosen action by generating Q-values indicating how good or bad it is in the state.

Asynchronous Advantage Actor-Critic (A3C) [66] is a actor-critic method that uses threaded workers to parallelize the training process. These workers use a global parameter network θ and θ^- where newly created workers can get their starting values from while better trained ones can update the network with their found policies. θ is used by the actor while θ^- is the target network used by the critic.

A synchronous, deterministic variant of the algorithm is A2C. A disadvantage of A3C is the asynchronous itself. If agents in parallel train they independently sync with the global networks. This means agents could work with outdated values for the parameters for a while, causing

Algorithm 6 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread [66].

1: *// Assume global shared θ, θ^-, and counter $T = 0$.*
2: Initialize thread step counter $t \leftarrow 0$
3: Initialize target network weights $\theta^- \leftarrow \theta$
4: Initialize network gradients $d\theta \leftarrow 0$
5: Get initial state s
6: **repeat**
7: Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
8: Receive new state s' and reward r
9: $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$
10: Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
11: $s = s'$
12: $T \leftarrow T + 1$ and $t \leftarrow t + 1$
13: **if** T mod $I_{target} == 0$ **then**
14: Update the target network $\theta^- \leftarrow \theta$
15: **end if**
16: **if** t mod $I_{AsyncUpdate} == 0$ or s is terminal **then**
17: Perform asynchronous update of θ using $d\theta$.
18: Clear gradients $d\theta \leftarrow 0$.
19: **end if**
20: **until** $T > T_{max}$

unstable training and a not so smooth convergence. If we let the agents communicate with each other to guarantee a synchronous update of the networks, the issue is resolved. This could be seen as a form of mini-batch gradient update since the number of steps for each agent is the same and a global average of the found gradients is used to update the global network.

2.2.2.3 Policy-based DRL: Proximal Policy Optimization

Proximal Policy Optimization (PPO) [67] is a gradient method that is derived from the *Trust Region Policy Optimization* algorithm (TRPO) [68].

The main idea behind TRPO is to constraint the size of the update to the policy by enforcing a *KL divergence*, which measures the divergence of one probability distribution from a second one [69]. It uses the following objective function that needs to be optimized [70]:

$$J_{(\theta)} = \mathbb{E}_{s \sim \rho^{\pi_{\theta_{old}}}, a \sim \pi_{\theta_{old}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\theta_{old}}(s, a) \right] \tag{2.24}$$

where $\pi_{\theta_{old}}$ is the behaviour policy for collecting trajectories and $\rho^{\pi_{\theta_{old}}}$ the discounted state distribution or the probability of visiting one state from another state according to the policy. $\hat{A}$ is the estimated advantage function, which is the estimated difference between the Q-value of a state-action pair and the value function of the state. It is an estimation because of the unknown nature of the true rewards. Maximizing the objective function has to be done in accordance to the *trust region constraint*, enforcing the distance between the old an new policy measured by the KL-divergence to be within parameter δ [70]:

$$\mathbb{E}_{s \sim \rho^{\pi_{\theta_{old}}}} \left[D_{KL}(\pi_{\theta_{old}}(.|s) \parallel \pi_\theta(.|s) \right] \leq \delta \tag{2.25}$$

ensuring the policy doesn't diverge too much.

The PPO algorithm simplifies TRPO by using a clipped surrogate objective with L_{CLIP} (shown in figure 2.5).

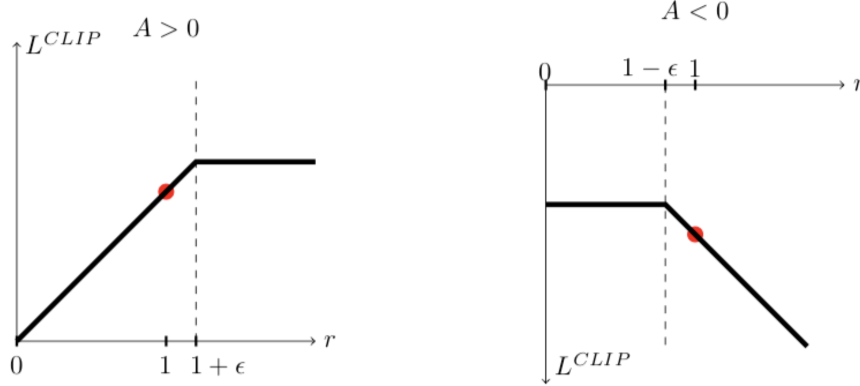


Figure 2.5: One single timestep visualized on surrogate L_{CLIP} in function of probability ratio r. The red dot represents the starting point for the optimization. The left curve shows a positive advantage, indicating a good action has been taken. The right one is of negative advantage, showing the impact of a bad action [67].

If we substitute $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ we can simplify the TRPO objective to:

$$J_{(\theta)}^{TRPO} = \mathbb{E}\left[r(\theta)\hat{A}_{\theta_{old}}(s,a)\right] \tag{2.26}$$

In TRPO we don't specify a limitation on the distance between θ_{old} and θ leading to instability when applying big updates to the parameters and high values for the ratio $r(\theta)$. PPO constraints the ratio to be a small interval around 1, defined in $[1-\epsilon, 1+\epsilon]$ with ϵ as a hyperparameter (not to be confused with ϵ from ϵ-greedy). The objective for PPO becomes:

$$J_{(\theta)}^{CLIP} = \mathbb{E}\left[\min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a))\right] \tag{2.27}$$

Because PPO can be implemented using several *workers* (processes that have their own threads to run on) a A2C inspired algorithm is given in the pseudocode.

Algorithm 7 PPO, Actor-Critic Style [67]

1: **for each** iteration= $1, 2, ...$ **do**
2:     **for each** actor= $1, 2, ..., N$ **do**
3:         Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
4:         Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
5:     **end for**
6:     Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leqslant NT$
7:     $\theta_{old} \leftarrow \theta$
8: **end for**

Chapter 3

Explainable AI

Before going forward with the subject of Explainable Artificial Intelligence (XAI), we first define two of the most important yet often falsely interchanged terminologies: *interpretability* and *explainability. Interpretability* is the degree to which a human can understand the outcomes of a system [8]. It is the ability to explain or to provide the meaning in understandable terms to a human [71]. Differently, *explainability* implies that the system provides an explanation of some kind why a certain outcome has been produced [9]. The term is linked to the notion of explanation as an interface between humans and a decision maker that is both an accurate proxy of the decision maker and comprehensible to humans at the same time [71]. According to Gilpin et al., who did a recent and comprehensive survey on the field, interpretability alone in an AI system is insufficient [72]. In order for humans to gain trust in *black box* models, insights into the process should be provided (providing interpretability). However, there is a need for the model to be completed with the capacity to defend chosen actions and provide relevant responses to questions from the user [72].

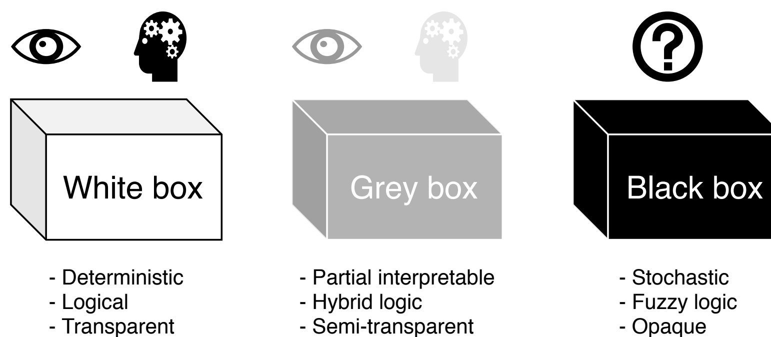


Figure 3.1: Three types of boxes for AI models, each given with characteristics. The darker we go, the less transparent and interpretable the model becomes [12, 11].

We briefly touched upon *black box* in the previous paragraph, which is one of 3 types of box models we can distinguish within ML. A black box model is a kind of machine learning model that is very hard to explain and to be understood by at least an expert in the practical domain [73, 11, 10]. Examples of these are ANNs and support vector machines (SVM) [74]. They are very powerful techniques yet their complexity doesn't allow for a clear view into their inner

workings. On the opposite side of the spectrum we have *white box* models. These are comprised of ML techniques inherently transparent and interpretable to a human user. They provide results associated to their models that are easy to understand by domain experts in the application domain [11]. ML types included are decision trees, k-nearest neighbours and linear regression. The terms understandable and interpretable are often used interchangeably in describing the properties of these kind of models. However their meaning differ since understandable models refer to ML that needs an additional model or technique to provide an explanation while an interpretable model can provide an explanation on its own [11]. Between black and white exits a grey area of box models. These *grey boxes* combine traits from both sides and are described as semi-transparent [75]. They use a priori knowledge concerning the process and unknown parts of the model are estimated using measured data [12]. This means only partial interpretability could be gained from these kind of boxes.

The first examples of XAI date back from the 70's [76, 77] where the main focus was on consultation systems. In earlier days when most expert systems were based on human-created rules in a knowledge base (KB), interpreting these models were accessible [78]. This kind of technique required the involvement of *domain experts* providing the necessary expertise to populate the database with relevant information. Next to KB systems a range of tree-based techniques were introduced with one of the most popular ones the decision tree [79, 80]. We will dedicate a further section on the subject of trees but note that their use could be dated as early as the 80's. In 1988, Michie proposes a criteria for AI to gain a level from weak to ultra-strong [81]. The ultra-strong criterion states that the system should be capable not only explaining how it has structured its acquired skills but also should be able to teach them. This is open for interpretation as it could mean the ability to teach to humans. The ambition to achieve this criterion is still relevant today [81]. From Swartout and Moore's review of expert systems (1993), we can summarize five desiderata for useful AI explanations [82]:

- **Fidelity**: the explanation must be a reasonable representation of what the system actually does.
- **Understandability**: terminology, user competencies, levels of abstraction and interactivity should be incorporated into the explanation.
- **Sufficiency**: the explanation should be able to explain function and terminology of the context. It has to be detailed enough to justify a made decision.
- **Low construction overhead**: providing explanations shouldn't dominate the cost of designing an AI system.
- **Efficiency**: the system shouldn't slow down significantly when a explanation needs to be given. This applies to both the AI itself as a potential sub-system providing the explanation.

These are only desiderata from one point of view as one can add many more like *universality*, meaning the agent doesn't rely on one particular language or dialect to communicate to the user, and *time awareness*, where the explanation can be different depending on time context and/or amount of seen training data. However, the summed up five points form a decent base on which an XAI architecture can be build on. With its rise in popularity starting in the early 2000's, deep learning brought forward inherently unexplainable black box models. It is a misconception that XAI started in this period, but in reality it gained a significant boost in popularity thanks to the unexplainable nature of ANNs [83]. A recent spike in interests came from the initiation of the DARPA's XAI program from 2017 [84]. With the program, the agency tries to improve explainability of AI in military applications.

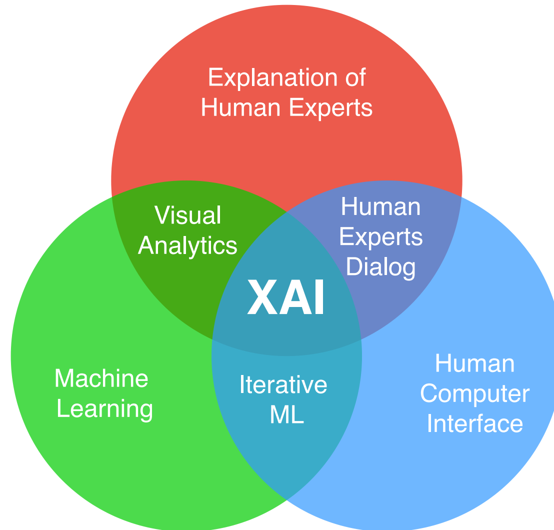


Figure 3.2: Interaction of different areas forming the origins of XAI as a field [9, 11].

The field of XAI is the culmination of three separate areas (figure 3.2). The field of ML contributes to the AI part of XAI. Human expertise explanations and human-computer interactions (HCI) contribute to the explainable part [9].

3.1 Motivations

Now that we know the origins of XAI, we have to motivate its usage and what purpose it can fulfill. Literature studies have underlined the importance of the purpose of explanations coupled with AI systems [85]. From a recent systematic literature review in XAI for robotics by Anjomshoae et al. we can list these motivations into seven categories [86]:

- **Transparency**: if the explanation increases the transparency of the model, a user can better predict what the model would recommend next. Humans have the ability to understand other human beings (and sometimes animals) but this assumption is not conform to complex mechanisms like computers and robots [87].
- **Trust**: human confidence isn't simple to gain. When the user (partially) knows how the agent thinks, the trust in it increases. This assumption is only accurate when the explanation facilitate the decision-making process. Explanations where ambiguity is present on how to act on the recommendation have no beneficial effect at all [88].
- **Collaboration**: with collaboration comes the benefit of increased efficiency and team performance. Sharing goals between users and agents increases the total work effort significantly more than when beliefs, the information about the environment which are perceived via the agents, are shared [89]. This is the main idea behind the *collaborative robots* movement or COBOTs where cooperation with robots rather then full replacement of human labour is proposed as a productivity enhancer [90].
- **Intent communication**: In situations where emphasis lies on human-agent interaction, intent communication is an important motivation in order to make the internal state of the

robot (its goals and intentions) understandable to humans [91]. When the agent assumes a leading role for the group, clear communication should convince the participating users that the system can provide competent instructions and strategies to manage the team.

- **Control**: as our military use case example implies, human should retain control over the AI by knowing what it is thinking about and to correct its policy where necessary. One should always consider which level of autonomy the system can be granted to based on the situation [92].
- **Education**: curiosity has lead many advances in sciences. Studying the behaviour of intelligent agents could lead to new insights in human an behavioural sciences [93]. These concepts could be used in education to learn more about robots and AI in general [94]. This educational aspect is also applicable for children [95] with the use of simplified and kid-friendly user interfaces like *Scratch* [96].
- **Debugging**: it is easier to debug or correct a system when the inner workings are known. For *Agent Oriented Programming* (AOP) one can already use debugging tools for intelligent systems [97]. This is done by 1) deriving the agent's choice of action from their beliefs and goals (providing reasons for doing something) and 2) evaluating the rule-based structure of these AOP applications to determine the program's behaviour [97].

From this review, they concluded that trust and transparency are the most prominent drives of the explanation in order to boost the user's confidence in the system [86].

3.2 Use cases

Nowadays, AI is used in applications from a variety of sectors (table 3.1). We analyse several use cases where XAI can bring added value to the product or service and its end user. Note that most examples are from situations where AI could have a noticeable impact when incorporated into the decision process [98]. The users effected should get insights in this impactful process in order for the system to motivate its findings.

The transport and logistics sector could benefit from AI with intelligent scheduling and fleet control systems based on multi-agent AI. Intelligent Transport Systems (ITS) have been expanded drastically due to cost-effective sensor networks and different kinds of distributed and cloud-based computing [99]. With the rise of the *internet of things* (IoT) and the promise of low latency 5G mobile networking, explainable ITS could be expanded upon to enhance and communicate the scheduling of the fleet to its stakeholders. The introduction of autonomous vehicles would require motivation how certain routes are driven and what happened inside the system's moments before a potential mayor failure like a crash [100].

A major use case of AI in healthcare would be the discovery of new drugs. Deepmind's *AlphaFold* is a DL based protein folding predictor that can help in the discovery of new structures usable in medical applications [101]. In the paper they remark the interpretation of the neural network and came up with a technique to understand how the inputs affect the final predictions. They used integrated gradients [102] to the distogram, which is a histogram of distances between the folds, to indicate the location of input features affecting the network's predictions of a certain distance [101]. Many treatment recommendation systems already exists [103]. However, the ones based on black box models still lack the interpretability of other used techniques [104]. A recent application, called *OnocoNetExplainer*, derives explainable predictions of cancer types based on ANNs trained on gene expression data [105]. With the use of visualisation of cancer-inducing

genes on a heatmap, medical consultation for a treatment could be more accurate. Then there is the usage of XAI for organizing population health and organization of medical institutions. These should accompany directing staff and politicians to decide the most effective policy.

AI systems are already involved in the U.S.A. for deciding the sentence of criminal offenders [106]. Most models have a bias towards race or gender, discriminating these groups with a higher chance on harder punishments [98]. Because of the great impact on the convict in question, one should review the decision of the system and let human intuition intervene with the strict interpretation of the law the system will rely upon. Intelligent systems are also applicable for big company mergers and acquisitions of rivals since the result could be disadvantageous for the consumer and the competition in that particular market. To prevent unfairness, a detailed motivation should be given with enough evidence to guarantee a healthy consumer market [107, 108, 109]. Since the introduction of the European *General Data Protection Regulation* act (GDPR) [110, 111], every EU citizen has the right to an explanation of an automated decision [107]. The challenge here is to provide an explanation to users not familiar with digital technologies and Internet [112].

The North Atlantic Treaty Organization (NATO) recently listed several possibilities and challenges for the use of AI in military applications [113]. Together with robust ML, against adversarial attacks [114], and data-efficient algorithms, AI should be sufficiently transparent and interpretable [113]. Major use cases are surveillance, underwater mine warfare, cyber security and many more. On the battlefield, autonomous drones could be guided by an AI which has also to decide which targets to engage with. This target analysis should be reviewed by a human controller to avoid unnecessary civilian casualties mistaken for enemy targets. For strategy recommendation, one should look no further than the movie *WarGames* form 1983 or the *Terminator* series with supercomputers *WOPR* and *Skynet* respectively trying to end all global military conflict by eliminating that what causes it: human nature. AI should definitely be hold back by at least several layers of human expertise before initiating an attack with a country's ballistic or nuclear arsenal.

Logistics [99]	**Healthcare** [104, 115, 116]	**Legal** [77, 107]
- Autonomous vehicles - Scheduling - Fleet control	- Drug discovery - Treatment recommendation - Population health - Organization planning	- GDPR - Criminal sentencing - Acquisition/merger evaluation
Defense [84, 117, 113]	**Financial** [118, 119, 120]	**Industry** [121]
- Target analysis - Autonomous drones - Strategy recommendation - Swarm control	- Risk management - Automated trade - Investment advice	- Infrastructure planning - Production line planning - Resource management

Table 3.1: Use cases for sectors with potential to benefit from XAI integration.

Many trades on the global stock markets are already automated by a system, possibly driven by AI [118]. Because of the huge volumes of stocks traded and the high financial risks involved, one should keep an eye on the portfolio not getting worthless by incompetent trades [122]. Risk analysis for credit and loans could be biased against certain ethnicities as shown in our legal example [120]. This is certainly preventable by introducing the human element. Like the medical centers, financial institutions can benefit from XAI powered operations and organisations [119]. These could boost the bank's efficiency in terms of processed customers and their satisfaction while also letting personnel participate in the planning by incorporating their suggestions into the model.

At last we have the industry sector which includes both primary and secondary industries. Before the production at a plant can begin, AI could help plan the infrastructure of the production line [123]. Explainable planning should comply to several demands in order to achieve an optimal infrastructure [124]. Besides a general motivation and a possible alternative to the chosen option, the system should explain what could be done to make the infrastructure more productive and efficient with the eye on achieving a higher *return on investment* (ROI). It also has to state why it couldn't get more efficient than a proposed configuration and whether or not a replan is necessary in the case of optimizing an existing production line. Automated resource management at the beginning of a line should communicate clearly and efficiently with suppliers to prevent a perforced halt on the line.

3.3 Performance-readability trade-off

Most literature describes an inherent tension between performance of AI, described in terms like accuracy, and explainability [84]. Apart from the performance-readability trade-off [125] several other terminologies are referred in literature, e.g. accuracy-comprehensibility [126], accuracy-interpretability [10] and performance-transparency trade-off [127]. To quote Breiman et al.: "accuracy generally requires more complex prediction methods ...[and] simple and interpretable functions do not make the most accurate predictors" [79]. In figure 3.3 classify several ML techniques based upon this trade-off. The simpler the model, the more interpretable and/or explainable it is. This however comes with a reduction in performance. The more complex the higher the accuracy but with lower comprehensibility as a model [128, 129]. This ranking follows the black-grey-white box spectrum as described before.

In a recent Nature article by Rudin, a motivation against such trade-off is given. She describes it as a mere misconception that there is necessarily a trade-off between accuracy and interpretability [10]. She concludes that there is often no significant difference between complex and simple classifiers after the data has been preprocessed in a correct manner Rudin. According to her, this is even true for applications like computer vision where black boxes have gained significant popularity over the years. An artificial trade-off can always be created by removing parts of a complex black box model, but this is not representative of the analysis such model would perform on a real problem [10]. Furthermore, she points to the fact that the existence of such a trade-off term has led many researches to forgo attempts to make interpretable models instead of greying out black boxes [10]. The graph in figure 3.3 gives a suggestive view of the relation between explainability and accuracy. It also lacks any measurement indication on the X- and Y-axis as well. However, Rudin is the only author from the list of researched publications who arguments against the trade-off and this purely based on her own work experience in XAI for healthcare and finance. This assumption has to be taken lightly for it doesn't share any similar ones from the scientific community.

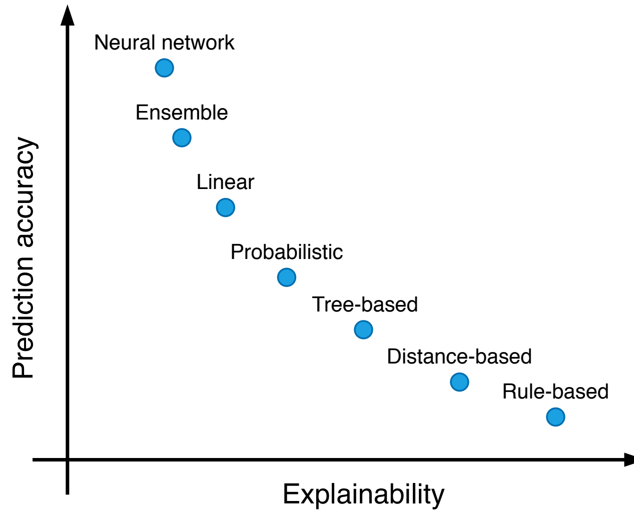


Figure 3.3: Visualisation of the performance-explainability trade-off from most performant, an ANN, to the most explainable one, rule-based techniques [107, 84].

3.4 Taxonomy

There are many categorizations of XAI methods within the domain. We studied overview and survey literature around XAI and interpretable AI in order to provide an analysis of the most common taxonomies used. We examined work by Molnar [8], Barredo Arrieta et al. [71], Guidotti et al. [130], Gilpin et al. [72] and Adadi and Berrada [128].

Figure 3.4 shows the used cross-referencing between the chosen publications. Note the youngest works are from 2018 while the most recent published ones are as recent as 2020. Molnar and Guidotti et al. form the base of this review while Barredo Arrieta et al. include all examined publications. We think the range of surveys chosen forms a comprehensive, yet diverse view on the current state of XAI as of 2020.

Molnar approached the terminology by going over three types of models roughly in accordance with the 3 types of box models (table 3.2). Firstly, transparent models can be classified as white box models. Secondly model-agnostic approaches try to grey out any type of uninterpretable model. Lastly, Molnar's example-based explanations category is applicable for black box models that have a secondary model capable of generating explanations of the inner decision making.

Similar to Molnar, Barredo Arrieta et al.'s taxonomy closely correlates to the different kind of box models (table 3.3). The difference here is Barredo Arrieta et al. don't provide a category for explanation generating methods. Instead they have two kinds of post-hoc methods, techniques applicable after the model has processed the dataset and output is given. Next to model-independent techniques, they include model-specific methods with examples from ANNs.

Guidotti et al. made a high level distinction between *reverse engineering* the black box model and *design* of explanations (table 3.4) [130]. The first one focuses on techniques for extracting usable information from the black box to construct answers why certain outcomes are predicted by the model. In the second case, given a dataset of training decision records a method creates an interpretable predictor model together with the corresponding explanation [130]. The difference between is the resulting product of the method. *Model explanation* generates a comprehensible

global predictor that mimics the behaviour of the original and provides explanations on the predictions made. *Outcome explanation* generates a local predictor that can mimic parts of the black box, imitating the behaviour on one single input, while providing an explanation of the prediction on that one single input.

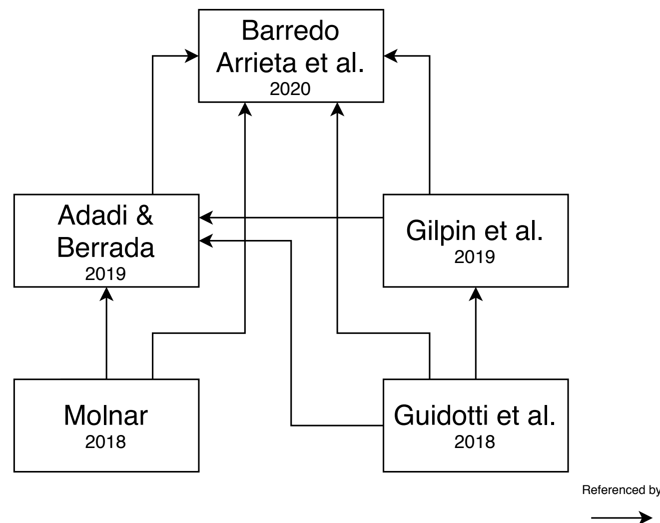


Figure 3.4: Dependency diagram of literature references. Arrow indicates the usage of one publication in another.

Once again, Gilpin et al. follows a similar taxonomy to Molnar (table 3.5). The difference here is the *processing* category where models are simplified rather than being inherently transparent. The *representation* category uses a so called *transfer task* to look into individual parts of the model.

At last we have the taxonomy from Adadi and Berrada (table 3.6). There approach follows a pseudo ontology to categorize XAI methods [128]. First one should decide if the technique has a global or local scope of explaining the model. For each kind of scope the model can be post-hoc and usually model-agnostic or it could be intrinsic which is by definition also model-specific. Examples of the *local intrinsic model-specific* category are not given.

As we can conclude, because of the diverse literature, there exists no conclusive taxonomy bringing all types of XAI methods together. Apart from the reviewed publications, other related work describes the methodologies in different terms. The categorization of Du et al. focuses more on the global or local post-hoc aspect similar to Adadi and Berrada [131]. Another used terminology is the one from Carvalho et al. that looks at when the explanation/interpretation takes place. This could be before (pre-model), during (in-model) or after (post-model) building the model [132]. The most extensive categorization would be one from Vilone and Longo who combined every possible aspect of a method into an comprehensive categorization tree as shown in figure 3.5 [133]. We encourage the reader to look beyond the scope of this thesis if more about XAI methods wants to be learned.

Molnar Interpretable Machine Learning. A Guide for Making Black Box Models Explainable. [8]		
Term	**Explanation**	**Examples**
Interpretable models	Models inherently interpretable by design.	- Linear regression - Logistic regression - Decision tree - Decision rules
Model-agnostic	Separating the explanations from the machine learning model.	- Feature importance - Feature interaction - Partial dependence plot (PDP) [134] - Individual conditional expectation (ICE) [135] - Accumulated local effects (ALE) plot [136] - Local interpretable model-agnostic explanations(LIME) [137]
Example-based explanations	Selecting particular instances of the dataset to explain the behaviour of ML models or to explain the underlying data distribution.	- Counterfactual explanations - Adversarial examples - Influential instances

Table 3.2: XAI taxonomy from Molnar. His division follows the different kind of box models a ML algorithm can be: white, grey or black box. This translates into fully, partially and zero interpretability given by the model.

Barredo Arrieta et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI [71]		
Term	**Explanation**	**Examples**
Transparent models	Models that convey some degree of interpretability by themselves.	- Decision tree - Logistic/linear regression - K-nearest neighbours - Rule-base learners - Baysian models
Model-agnostic (post-hoc)	Techniques designed to be plugged to any model with the intent of extracting some information from its prediction procedure.	- Saliency - Sensitivity - Influence functions - Shapley plots
Model-specific (post-hoc)	Post-hoc methods designed for specific types of models.	- Activation clusters - Loss modification - Layer modification - Model combination - Feature extraction - Caption generation

Table 3.3: XAI taxonomy from Barredo Arrieta et al.. This follows a interpretable, semi-interpretable scheme with the latter one being possible for model independent as well as model specific post-hoc techniques.

Guidotti et al. A survey of methods for explaining black box models [130]		
Term	**Explanation**	**Examples**
Transparent design	Providing a model which is locally or globally interpretable on its own.	- Rule set [138] - CPAR [139] - 1Rule [140] - IDS [141]
Black box inspection	Providing a representation for understanding either how the black box model works or why the black box returns certain predictions more likely than others.	- Prospector [142] - Auditing [143] - TreeView [144] - OPIA [145] - NID [146]
Model explanation	Providing an interpretable and transparent model which is able to mimic the behavior of the black box and which is also understandable by humans.	- Tree metrics [147] - Conjunctive rules [148] - GoldenEye [149] - Trepan [150] - TSP [151]
Outcome explanation	Providing an interpretable outcome, that is a method for providing an explanation for the outcome of the black box.	- LIME [137] - MES [152] - CAM [153] - Grad-CAM [154]

Table 3.4: XAI taxonomy from Guidotti et al.. This taxonomy follows the different type of box models but with two type of explanatory categories.

Gilpin et al. An overview of interpretability of machine learning [72]		
Term	**Explanation**	**Examples**
Processing	Minimize the complexity of explanations (essentially, minimize length) as well as local completeness (error of interpretable representation relative to actual classifier, near instance being explained).	- Proxy methods - Decision tree - Salience mapping - Automatic-rule extraction
Representation	Characterize the role of portions of the representation by testing the representations on a transfer task.	- Role of layers - Role of neurons - Role of vectors
Explanation producing	Generate human-readable explanations that can be tested by similarity to test sets, or by human evaluation.	- Scripted conversations - Attention-based - Disentangled representation - Human evaluation

Table 3.5: XAI taxonomy from Gilpin et al., also largely following the box models.

Adadi and Berrada Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI) [128]		
Term	**Explanation**	**Examples**
Local intrinsic model-specific	Explaining the reasons for a specific decision or single prediction for a specific model.	/
Global intrinsic model-specific	Understanding of the whole logic of a specific type of model and follows the entire reasoning leading to all the different possible outcomes.	- Decision tree - Rule lists
Local post-hoc model-agnostic	Understanding the local inner workings, not tied to a specific model.	- Counterfactuals explanations - Decomposition - Feature importance - Saliency map - Shapely explanations - PDP [134] - Rule extraction - LIME [137]
Global post-hoc model-agnostic	Understanding of the global inner workings, not tied to a specific model.	- Feature importance - Model distillation - PDP [134] - Rule extraction - Activation maximization

Table 3.6: XAI taxonomy from Adadi and Berrada. Their categorization is mainly based on the division whether a technique is considered model-specific or model agnostic and what the scope of interpretability is.

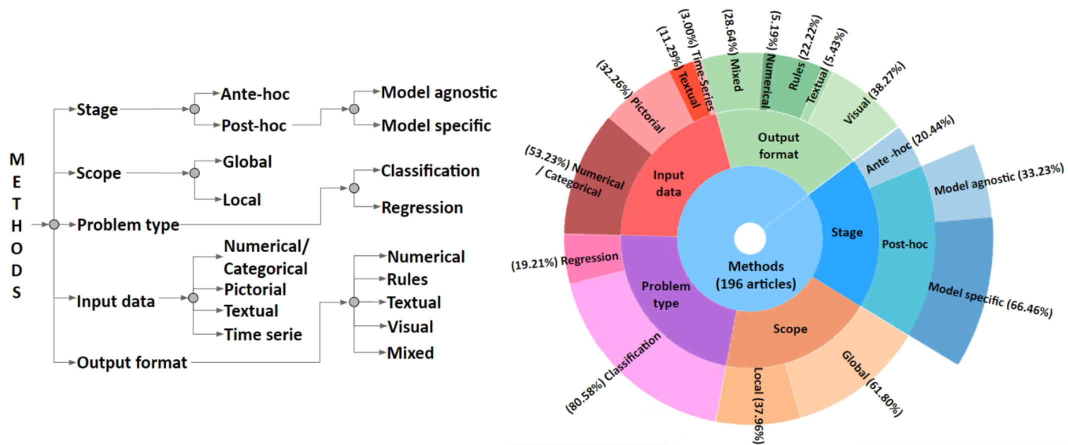


Figure 3.5: Vilone and Longo's categorization tree for XAI methods (left) and distribution of articles across categories (right) [133].

XAI categories		
Inherently interpretable models	**Post-hoc representations**	**Additive explainable models**
White box models who's inner workings are interpretable by a human.	Grey box techniques giving insight into parts of the model by visualisation or other forms of information extraction.	Secondary mechanisms able to generate human readable explanations from a black box model.
- Interpretable models [8] - Transparent models [71] - Transparent design [130] - Processing [72] - Local intrinsic model-specific [128] - Global intrinsic model-specific [128]	- Model-agnostic [8] - Model-agnostic (post-hoc) [71] - Model-specific (post-hoc) [71] - Black box inspection [130] - Representation [72] - Local post-hoc model-agnostic [128] - Global post-hoc model-agnostic [128]	- Example-based explanations [8] - Model explanation [130] - Outcome explanation [130] - Explanation producing [72]
- Decision trees - Rule lists - Bayesian models - K-nearest neighbours - Logistic/linear regression	- Saliency maps [1] - Activation maximization - Rule extraction - Model distillation [3] - LIME [137] [3]	- Adverserial examples - Counterfactual examples [2] - Scripted conversation - Disentangled representation - Conjunctive rules

Table 3.7: Our categorizing taxonomy for the types of XAI approaches.

In order to achieve a summarization of all taxonomies, we propose our own taxonomy which respects the reasoning of each categorization as much as possible (table 3.7). We present our taxonomy together with definitions, example techniques and corresponding taxonomy from literature. We based the categorization mostly on Molnar's work because of the link with the sort of box models available. The first category are the *inherently interpretable models*. These include all white box models as well as intrinsic model-specific methods because of their interpretable nature. The second category, *post-hoc representations*, corresponds more with grey box methods in using post-hoc methods. They provide insights into parts of the black box by using several techniques like rule mining and visualisation methods. All model-agnostic, post-hoc methods, representation and black box inspection methods are part of this category. *Additive explainable models* are all secondary model techniques that can produce an explanation given a black box

[1]Technique could be classified as *inherently interpretable model.*
[2]Technique could be classified as *post-hoc representation.*
[3]Technique could be classified as *additive explainable model.*

as input. All techniques that generate an explanation are categorized under this term. Note that several techniques like LIME and saliency maps are examples we can find in a different category than represented in this taxonomy. These are open to interpretation but are categorized according to the original literature they came from.

3.5 Conventional techniques

For each of our taxonomy terms, we discuss an example technique relevant to the further scope of this thesis. We begin by discussing methods well-known inside the XAI community.

3.5.1 Inherently interpretable model: decision trees

Classification and Regression Trees (CARTs), commonly called *Decision Trees* (DT), are tree structured AI techniques capable of both classification and regression tasks [79]. They are ML methods for learning discrete-valued target functions and are one of the most popular inductive inference algorithms in use [4].

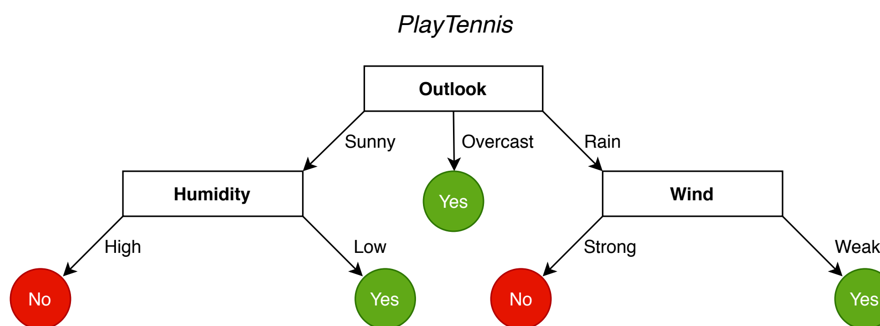


Figure 3.6: A decision tree example for the concept *PlayTennis* which answers whether or not tennis can be played according to the weather [4, 80].

Most often a DT is represented as a binary tree, asking a binary question in each of its nodes. This can also be extended to a n-ary tree with at most n possible directions per node. The structure consists of three types of nodes. The *root* node forms the start of the tree and has no parent nodes above it. *Leaf* nodes form the bottom of the tree and have no children of their own. These nodes will answer to the question the DT is trained on. All other nodes are called *internals* and have both a parent and several children. Instances in the dataset are sorted down the tree from the root down to the a particular leaf node that makes the final decision for the instance [4]. While traversing down towards a leaf, each internal node encountered will be compared against the attribute specified by this node in order to know which direction should be descended to. The possible decisions the structure can make are ranges of values in the case of regression or a single class(es) prediction.

Most algorithms developed for learning DTs are variants on a core algorithm that uses a top-down greedy search through the space of possible DTs [4]. A well known basic algorithm is Iterative Dichotomiser 3 (ID3) [80], shown in algorithm 8. This algorithm learns by constructing the tree top-down by looking at which attribute should be tested at he root. A statistical test determines the likelihood of being the one that classifies the training examples the best. The best attribute is selected and used as the test at the root [4]. One descendent per possible value

of this attribute is created and the process repeats itself for every resulting child node with the appropriate sorted set of training examples. This is done until all attributes are represented by a node so a decision path can be made from the root to a leaf node.

To define the best classifying attribute in a node, one can calculate *information gain* of selecting that particular attribute. The attribute with the highest gain would best split the dataset at that stage of the learning process. To calculate *gain*, we have to first calculate *entropy* [155], measuring the homogeneity or purity between examples. The entropy $H(x)$ of examples x relative to a classification with c possible labels is defined as in equation 3.1. Here p_i is the ratio of examples similar to i of all examples x. For a binary case this is $H(x) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$ with $p_{\oplus}$ the probability of encountering a positive example in x and $p_{\ominus}$ a negative example (figure 3.7). The highest *gain* is defined by (3.2), where A is an attribute of the set. *Gain* measures the expected reduction in entropy.

Algorithm 8 ID3(*Examples, Target_attribute, Attributes*)

1: Create a *Root* node for the tree
2: If all *Examples* are positive, return the single-node tree *Root*, with label = +
3: If all *Examples* are negative, return the single-node tree *Root*, with label = -
4: If *Attributes* is empty, return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
5: Otherwise:
6: **begin**
7: $A \leftarrow$ the attribute from that best[1] classifies *Examples*
8: The description attribute for *Root* $\leftarrow A$
9: **for each** possible value, v_i, of A **do**
10: Add a new tree branch below *Root*, corresponding to the test $A = v_i$
11: Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
12: **if** $Examples_{v_i}$ is empty **then**
13: Below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
14: Else below this new branch add the subtree $ID3(Examples_{v_i}, Target_attribute, Attributes - \{A\})$
15: **end if**
16: **end for**
17: **end**
18: Return *Root*

$$H(x) \equiv \sum_{i=1}^{c} -p_i log_2 p_i \tag{3.1}$$

$$Gain(X, A) \equiv H(X) - \sum_{v \in Values(A)} \frac{|X_v|}{|X|} H(X_v) \tag{3.2}$$

Like other ML algorithms, DTs are prone to overfit. This depends on when the learning process stops and post-processing in the form of pruning the tree. Overfit occurs because of their data intensive nature, meaning the data is tested extensively on each possible attribute per node. At every node the structure looks at every possible split of every independent variable.

[1] The best attribute is the one with the highest gain in information according to (3.2).

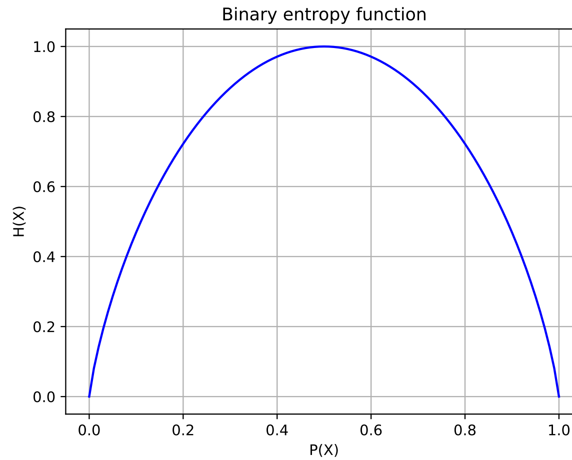


Figure 3.7: Binary entropy function, $H(x)$, relative to a boolean classification. The proportion of positive examples, $P(x)$, varies between 0 and 1 [4].

Early-stopping [156] can prevent the grow of the tree when overfit starts to occur when an increase in test loss becomes present. Overfit could also be regulated by post-processing the structure. One applicable technique is *pruning* of the DT. With pruning, parts of the tree are pruned down to decrease the tree's complexity so a simpler hypotheses could result from it [80]. According to *Occam's Razor*, simpler explanations are most often the better ones compared to more extensive ones. Many types of pruning are possible [157], but the most basic is the *reduced-error pruning* [158].

3.5.2 Post-hoc representation: feature visualisation

Two possible approaches could be considered when observing a black box model. First we can view the model on a global scope, observing its behaviour for a given input. As a second method we could zoom in on individual parts that make up the model and see interactions with *neighbouring* components or on a individual level. For a more local view we could investigate parameters like weights and biases in a neural network. The weight of a neuron can tell us about the importance of that neuron when activated by a part of the input. Visualizing the weights in a *heatmap* can gives us meaningful insight in how certain regions of the input are more impactful for a decision by the layer than others [159].

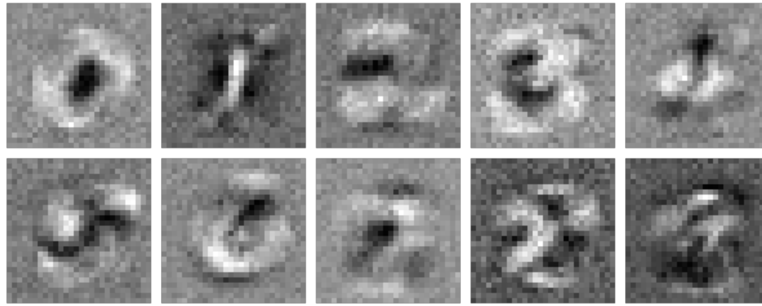


Figure 3.8: Visualizing the weights for all the output neurons of an MNIST classifier [160].

We give an example for a simple feedforward ANN classifier with one hidden layer, trained on the MNIST dataset. As can be seen in figure 3.8 the visualised weights resemble the digits or classes from 0 to 9 indicating the role of a output neuron during classification. The lighter the region, the higher the weights in that area. When most of the activation happens in this lighter region (meaning the input mostly corresponds with these regions), it is an indication of having that particular node deciding the outcome class. When applied on other datasets, the visualisation of FNN weights could be insufficient. In figure 3.8 we see the weights visualisation of a classifier trained on CIFAR-10, an image dataset with 10 classes. The input differs from MNIST in the dimensions (32x32 instead of 28x28) but also in the amount of color channels (3 for RGB instead of 1 for greyscale). Because of the added complexity, the resulting heatmaps of the neurons are far less interpretable. Some features we could recognize are the blue sky of the airplane class and water of the ship class [160]. Also green for the frog class is distinctive.

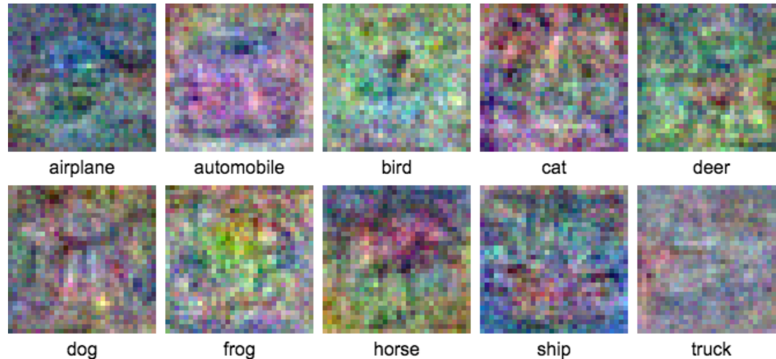


Figure 3.9: Visualizing the weights for 1-layer CIFAR-10 classifier [160].

When training CNN classifiers, other components could be visualised such as the learned feature maps [161]. These are the resulting activations when using a kernel filter on the previous layer input. Kernels can also be visualised as seen for the CIFAR-10 dataset in figure 3.10 but often lack meaningful context without the original input.

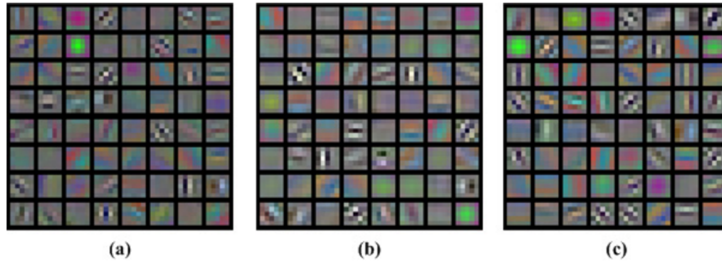


Figure 3.10: Visualization of 64 features learnt in the first convolutional layer on the CIFAR-10 dataset. Several different pooling techniques were used to learn the features. The size of each feature is 553. (a) Features learnt with max pooling. (b) Features learnt with average pooling. (c) Features learnt with mixed pooling [162].

3.5.3 Additive explaining model: rule list

Rule learning is one of the oldest fields of machine learning [163]. Rule-based ML techniques have proven their use in both propositional learning and relational learning and could be integrated into a wide range of applications. Because of their self-explanatory composition, a list of decision rules, the interpretation is as straightforward as reading these in the correct order. A decision rule is a simple IF-THEN statement consisting of a *condition* (IF) and a *prediction* (THEN) [8]. Conditions can be joined together with a conjunction (AND), making the condition more specific.

$$\texttt{IF}\ \mathit{Conditions}\ \texttt{THEN}\ c \tag{3.3}$$

where c is the class label, and *Conditions* are a conjunction of simple logical tests describing the properties of the instances that have to be satisfied for a certain rule [163]. These propositional rules are learned by a *induction* algorithm, of which many exist. Several well-known are CN2 [164], RIPPER [165], and PRIM [166]. We focus on the CN2 algorithm as recently done work by Coppens et al. incorporates the algorithm in the context of *Relational Reinforcement Learning* (RLL), which is an interpretable RL approach.

Recently a policy distillation algorithm has been developed based on the CN2 rule mining algorithm to distill a DRL policy into a rule-based decision system [167]. Apart from the use of RRL to match performance of the original policy, the contribution added *meta-information* into the formation of the rule list. This allows for choice within the rule miner by having near-optimal actions for a certain state [167]. The second made contribution is a new two-step approach identifying *important* misclassifications leading to shorter rule sets [167].

3.6 Knowledge distillation

We introduce several emerging methods within the field of XAI. We start by discussing *knowledge distillation* which is a technique that can be used to transfer the behaviour of one model into another surrogate one. Afterwards we propose two types of tree models that are used in the context of this thesis: *Soft Decision Tree* and *Adaptive Neural Tree*. These *neural trees* try to unify both the deep learning and the tree based learning paradigm by using a decision tree where the decisions inside the nodes are made by small neural networks.

When deciding upon the architecture of a deep neural network a range of different parameters should be considered during the design process. Hyperparameter values defining the number of layers of a network and how these are comprised out of nodes are not chosen straightforward and should be estimated via a machine learning modeling process. This approach considers different kind of model classes and their configuration (the structural identification) as well as the settings of the inner components like weights and biases (the parametric identification). The class of network defines the values needed for parametric identification. Aside from the composition of feed-forward network (FNN) layers, one has to specify the kernel size in the convolution layers of an convolutional neural network (CNN) and the type of memory cells used in recurrent neural networks (RNN). At last model cross-validation is used as a measure of performance of a chosen layout, eventually choosing the one that gains lowest value in a given cost function like mean squared error (MSE). Overall, finding the best suitable model depends on a computational intensive search on both the composition of the model as well as the settings of its parts where for each considered model we have to train it and validate using a given data set.

A way to omit this intensive search is the use of many different yet not high accurate predictors and compute the average of their combined predictions [168]. This *ensemble* method has proven to increase the accuracy of models significantly while effort needed for individual training is greatly reduced. The individual models themselves don't have to gain high accuracy in their task. Even from the aggregation of models that are slightly more efficient than a randomized one can a good predictor emerge [169]. However, the trade off to be made for an increase in accuracy is a higher amount of memory needed to store the models. In addition on large scale the whole ensemble of these cumbersome models can become computationally expensive, undermining the benefits of the performance gain. In the past several approaches have been considered with the goal of reducing the size of these ANNs. Starting in the late eighties, Hanson & Pratt's *network pruning* was among the first techniques that tried to compress the size of an ANN model [170]. With the usage of a biased weight decay at each training step, the proposed method insures the nullification of small weight in the network over time while retaining the larger weights. If uniform weight decay should be considered it would result in the same decay rate for both high and low weights. As a result of the biased version more of the model's input is condensed into a smaller number of weights, hence the term *minimized networks* they use.

In 2006, Bucil et al. propose a method based upon the generation of a new set of examples by the original model [171]. This approach was mainly targeted towards the reduction of large ensembles of deep learning models but is also applicable to other classes of models. The main idea behind this technique, called *network compression* is to have a cumbersome model generate pseudo training data from a large data set of unlabeled instances. This set can then be used to train a smaller model that will learn how to predict like the original one. The predicted labels are used as targets during the training of the second model. The newly generated data represents how the initial model correlates the input with the output without having knowledge about its inner workings. Several strategies can be applied when generating the pseudo data set. The

most straightforward one is to randomly sample values from the marginal distribution of each attribute. This non-parametric bootstrap approach however considers the entire possible input space, resulting in a lower focus on the important regions. Another technique is to estimate the underlying joint distribution of the attributes. This creates samples that maintain the conditional probabilities of the domain, representing relevant parts of the input space. These probabilities can be estimated with an mixture model algorithm which relies on a mixture of components with distinct distributions. Naive Bayes Estimation (NBE) [172] is such technique applicable on both discrete and continuous values of the attributes. The drawback of NBE and mixture model algorithms in general is the increase in computational complexity when large amounts of attributes are considered. Bucil et al. developed a new sample technique called MUNGE which uses a non-parametric estimate to estimate the joint distribution [171]. MUNGE chooses for each entry in the distribution the nearest neighbour (NN) by finding the shortest euclidean distance between both entry's attribute values. For each attribute there is a probability p that a change will occur. In the case of a discrete variable, we swap the value with the corresponding value of the NN. In the continuous value case, the attribute is normalized in $[0, 1]$ a random value is taken from a normal distribution with the NN's attribute as the mean and the distance between as the standard deviation. The number of iterations over all elements in the training set k depends on size of the pseudo data set needed to be generated. Every iteration uses the original data set and the result will be the unification of the k sub sets generated by the algorithm. Tests have concluded that MUNGE outperforms both RANDOM and NBE [171]. To train the smaller model on the pseudo dataset generated by the cumbersome model a MSE cost function is minimized between the logits, the final inputs to the softmax layer of the model.

In 2016, Hinton et al. introduce *Knowledge distillation*, a more generalized form of model compression. Bucila's motivation for the use of logits (q) instead of the softmax produced probabilities (z) (called *soft targets*) is the low influence they have on the cost function. To solve this, Hinton introduced the concept of a temperature variable T in the softmax function, regulating the exp function's influence on the logits. As seen in *equation* 3.4 a low temperature generates harder targets while high temperatures generate higher ones.

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3.4}$$

Aside from the use of soft targets, *knowledge distillation* uses a weighted average of two cost functions. The first one is the cross-entropy between the logits of the large and small network at the same high temperature as when the pseudo data is generated. The second one is cross-entropy between the correct labels of the large model and those predicted by the distilled version at an temperature of 1. Experiments have shown that the latter is less important and should be weighted with a lower value.

Further work in the field concerns techniques to determine the architecture of the distilled model [173] [174]. Because *knowledge distillation* isn't the focus of this thesis but rather a tool to incorporate explainable AI, we will not focus on these.

Distillation into other classes of models

As noted before, *knowledge distillation* allows us to train other classes of models different from the original one. This provides us the flexibility of choosing a better suited model for desired improvements in accuracy or readability. In a previous section we mentioned classes that are more human interpretable than black box models like neural networks. In the following sections we introduce two types of tree-based models where KD can be used with.

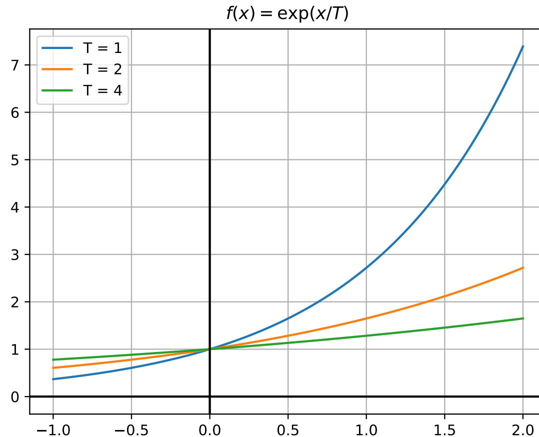


Figure 3.11: When increasing temperature T the exponential function doesn't rise in the same way as for lower temperatures.

3.6.1 Soft decision tree

A soft decision trees (SDT) is a variant of decision tree where all child nodes are selected with a certain probability rather than using hard internal nodes [15, 175] (figure 3.13). This means that while traversing the tree all possible paths towards the leafs will be considered since every leaf contributes to the final decision with a different probability. Different implementations of SDT exists in literature. We will use the one introduced by Frosst and Hinton throughout this thesis.

Each inner node i of the SDT has a learned filter $\mathbf{w}_i$ and a bias b_i [175]. Each leaf node l has a learned distribution Q_l for making a decision on the given input $\mathbf{x}$. The routing function, deciding at each internal node what the probability is of taking the rightmost child, is defined as:

$$p_i(\mathbf{x}) = \sigma\left(\mathbf{x}\mathbf{w}_i + b_i\right) \tag{3.5}$$

with σ being the sigmoid logistic function. This model is comparable to a *hierarchical mixture of experts* [176] with the difference of each expert being a so-called *bigot*, which is another word for "intolerant" leafs. A bigot doesn't look at the data after training instead it always produces the same distribution over the possible outcome classes. A hierarchy of filters is learned by the model where each bigot is assigned to a particular path probability following the branches from root to bottom. Each leaf learns a static softmax distribution:

$$Q_k^l = \frac{\exp(\phi_k^l)}{\sum_{k'} \exp(\phi_{k'}^l)} \tag{3.6}$$

where k is the number of possible output classes, Q^l denotes the probability distribution at the lth leaf with ϕ^l the parameters of the leaf.

Decisions in a SDT can become too "soft" during training. To prevent this, a temperature β is introduced to regulate filter activations prior to calculating the sigmoid. The probability from (3.5) becomes $p_i(\mathbf{x}) = \sigma(\beta\left(\mathbf{x}\mathbf{w}_i + b_i\right))$ at an internal node i. This parameter value is learned by the model during the training process.

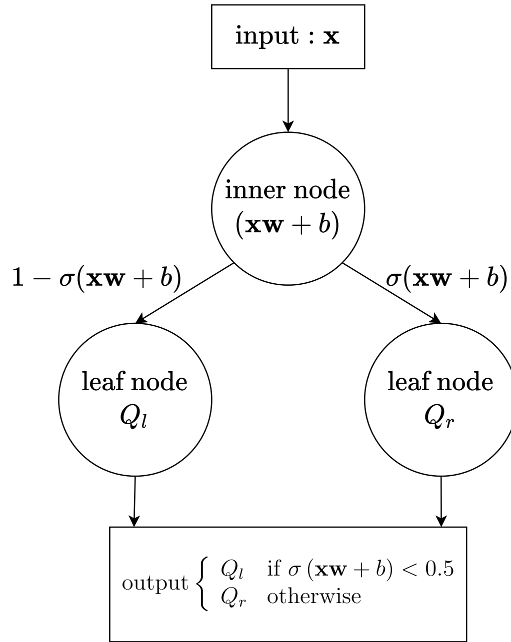


Figure 3.12: Diagram of a soft binary decision tree with a single inner node and two leaf nodes [175]. Temperature β is not included in the calculations.

There are two ways an SDT can make predictions. It could use the distribution of the leaf with the greatest path probability or it could average the distributions over all leafs weighted by their reaching probabilities. In the first case, a single path of encountered filters would result in a single distribution to be chosen. When taking all paths into account, an increase in accuracy would follow as a result. However, this comes at the cost of being more computational resource demanding since all filters of the tree have to evaluate their gating function.

SDTs have already been proven to be a suitable model for *knowledge distillation* by Frosst and Hinton. They were able to create a MNIST classifier from a SDT of depth 4 and arity 2 with a deep neural network as the model to distill from. Figure 3.14 shows the visualisation of the tree along with the used filters in the nodes to split the dataset. If we look at the rightmost internal node, parent to leaf nodes labeling the input as 3 or 8 and indicated in red in figure 3.14, we can see that the white area is important in deciding one of those two digits. If the area is connected the structure classifies it as 8 but if it is open it would predict 3. It is such indications a model should provide in order to better explain a particular path towards prediction a model makes. Note however that the closer to the root and due to the small arity, the explanation why a certain side is chosen by the routing function is not trivial. If we want to classify a 0, we have to start on the left side of the root. However, no distinct indication of the number is present in the visualisation of the root nor the child node. It is only when we arrive at the last parent node, chosing between 0 and 5, that we clearly can see a zero in the white of the visualisation.

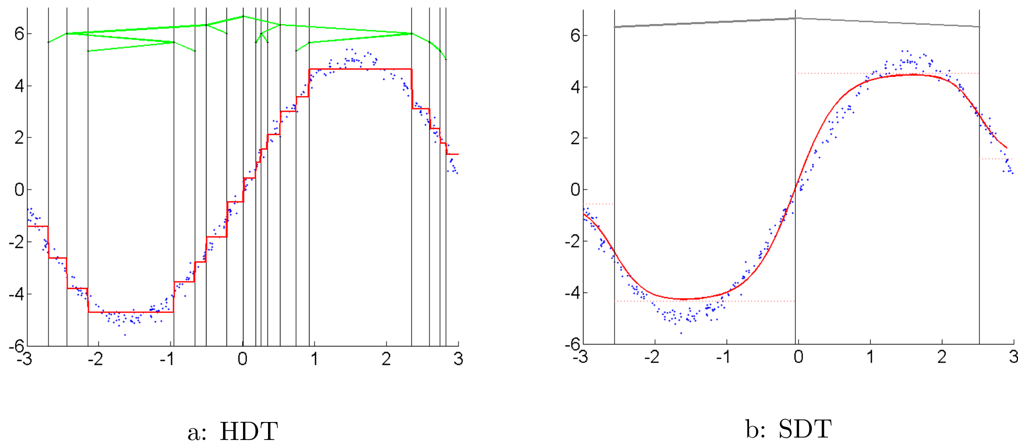
3.6.1.1 Training a SDT

When initializing a SDT before training, the structure is loaded with a certain depth d into memory. For a binary tree with arity 2 the amount of nodes is equal to $2^d - 1$, which results

in a structure with 2^{d-1} leafs and $2^{d-1}-1$ internal nodes. Parameters are arbitrary set at the beginning of the learning process. The goal is to minimize the loss function given the predefined structure and on a certain dataset D with $\mathbf{x}_j \in D$ being an input vector from the dataset. For a given target distribution T, the used loss is the cross entropy between each leaf (weighted by its reaching probability) and T. This is formulated as:

$$L(\mathbf{x}) = -\log\left(\sum_{l \in LeafNodes} P^l(\mathbf{x}) \sum_k T_k \log Q_k^l\right) \tag{3.7}$$

where $P^l(\mathbf{x})$ the probability of arriving at a leaf l with given input $\mathbf{x}$.



a: HDT

b: SDT

Figure 3.13: Comparing the resulting interpolation (red curve) between hard and soft regression. A smoother plot is generated in the SDT case with less splits (black vertical lines) given the same dataset (blue dots, sampled from a sinusoidal with Gaussian noise) [15].

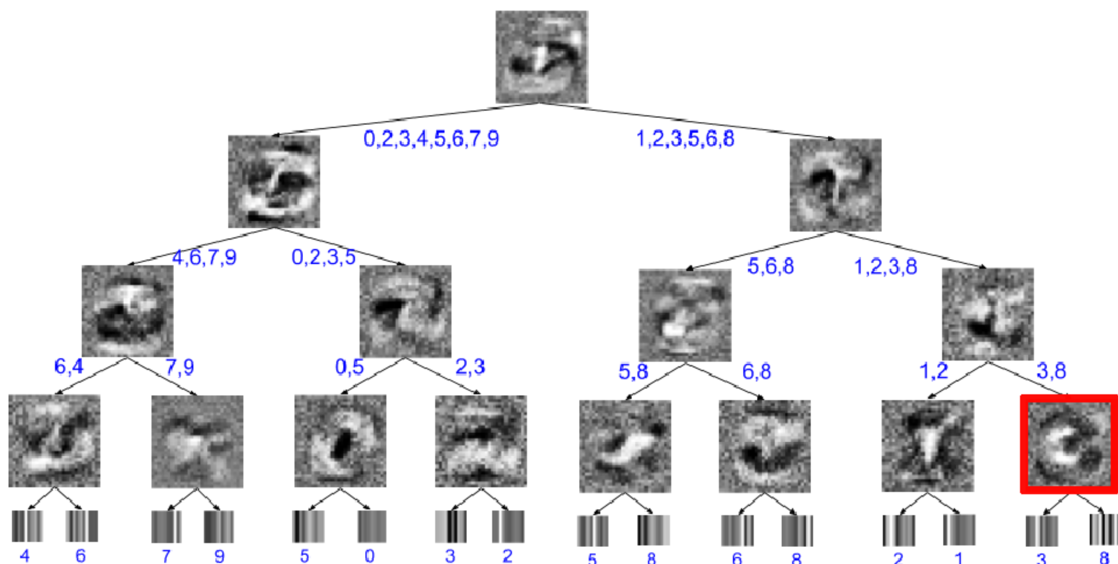


Figure 3.14: Visualization of the distilled SDT MNIST classifier. The nodes represent the used filters on which the splits are based upon [175].

A difference with *hard decision trees* (HDTs) is that SDTs use mini-batch gradient descent to optimize all parameters simultaneously instead of using a dynamic growing structure where splits are made one node at a time [177].

To encourage each internal node to use both their respective subtrees equally, a regularization cost C is introduced. This is based on the cross-entropy between the current distribution of a branching node α_i, formulated as:

$$\alpha_i = \frac{\sum_{\mathbf{x}} P^i(\mathbf{x}) p_i(\mathbf{x})}{\sum_x P^i(\mathbf{x})} \tag{3.8}$$

and the discrete binary uniform distribution:

$$C = -\lambda_i \sum_{i \in InnerNodes} 0.5 \log(\alpha_i) + 0.5 \log(1 - \alpha_i) \tag{3.9}$$

where $P^i(\mathbf{x})$ is path probability from root to node i and λ_i a hyper-parameter determining the strength of the penalty at node i. An assumption that could be made is that a tree uses alternative sub-trees equally and that this would be better suited for a classification task. However, this is less true the deeper one goes into the tree where a penultimate node would only responsible for two classes of input [175]. In non-equal proportion this would have the effect of penalizing the node for a non-equal split, decreasing the accuracy of the model. This is why a decaying penalty cost λ is used, which starts with value λ and is decayed over the tree depth by a factor of 2^{-d} [14].

3.6.2 Adaptive Neural Tree

One of the disadvantages of SDT's is the possibility of the gradient descent optimization to get stuck in a local minimum [15]. This however can be solved with the introduction of noise in the form of a random step in the descent algorithm. Another disadvantage is their fixed depth,

arbitrarily chosen when implemented. When chosen too small, the tree wouldn't perform well. When chosen too large, human interpretability will suffer. In a regular, not-optimized decision tree both under- or overfitting can occur when the depth of the tree is too low or too high respectively. Finding the optimal depth of a tree is not trivial. Nodes at a low level only use a small fraction of the training set causing the tree to overfit. When possible an exponentially large dataset in relation to the tree depth can prevent this [175].

Commonly used algorithms consider a top-down approach when deciding where to split the dataset. Because each split is made independently from possible splits further down the tree, the final resulting tree could be one unable to fully grasp the underlying characteristics of the given training dataset, lowering performance on new data samples that were never encountered before [178]. A frequently used strategy is to grow a tree in a top-down way as deep as possible after which the tree gets pruned, decreasing complexity and preventing potential overfit. This method however could waist computational resources as a large portion of the structure could be removed during the pruning phase. One type of architecture able to dynamically grow is the *Adaptive Neural Tree* (ANT) [18]. The algorithm uses several regulators to optimize the structure in both depth and arity.

An ANT is defined as a pair $(\mathbb{T}, \mathbb{O})$ with $\mathbb{T}$ defining the topology of the model and $\mathbb{O}$ the set of possible operators possible on the topology [18]. In the original paper the topology is an instance of a binary decision tree but this can be expanded upon to be a tree of any arity as long as the used routing in the nodes is capable to support this. A node can be one of three kinds: an internal node, a leaf node or a single parent node to a leaf node. The topology itself is denoted by $\mathbb{T} := \{\mathcal{N}, \mathcal{E}\}$ where $\mathcal{N}$ contains all nodes of the topology and $\mathcal{E}$ the edges connecting the nodes. In ANTs $\mathcal{E}$ contains an edge connecting the input data of the model to the root node via a transformation, which is different from a standard DT where the root routes the data first.

There are three types of operators contained in $\mathbb{O}$. These modules are routers $\mathcal{R}$, transformers $\mathcal{T}$ en solvers $\mathcal{S}$:

Routers $\mathcal{R}$
Every internal node of the tree contains a router $r_j^\theta \in \mathcal{R}$. Here θ contains the parameters of the routing function, mapping the input $\mathcal{X}_j$ at node j to $[0, 1]$. These modules contain the necessary computations to decide which edge the input should follow down the tree towards the next child. An example of a router is a small neural network.

Transformers $\mathcal{T}$
Every edge of the tree has one or several transformer modules. Each transformer $t_e^\psi \in \mathcal{T}$ applies a nonlinear function, parameterized by ψ, to the input it receives from previous modules. Unlike DT's, the edges transform the data and are allowed to grow by adding more operations, learning deeper representations [18]. A convolutional filter could be applied as an example for a transformer.

Solvers $\mathcal{S}$
Each leaf node $l \in \mathcal{N}_{leaf}$ has an assigned solver $s_l^\phi : \mathcal{X}_l \rightarrow \mathcal{Y} \in \mathcal{S}$ with parameters ϕ. Solvers receive transformed input data from the transformers on the edge leading to that solver. They output an estimate on the conditional distribution $p(y \mid x)$. For example a bigger neural network, such as a CNN, could be used as a solver.

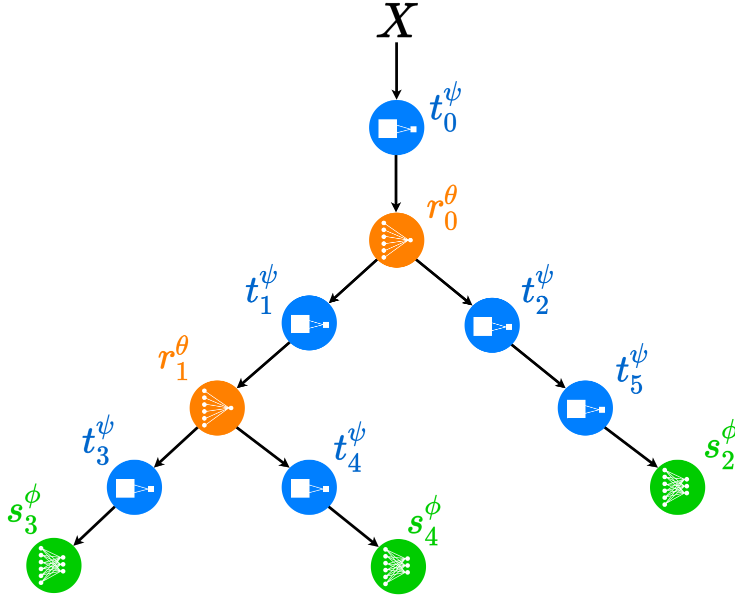


Figure 3.15: Visualisation of an ANT with its components. Blue nodes represent transformers, orange ones the routers. Solvers are indicated in green. X is the given input to the tree. [18]

3.6.2.1 Training an ANT

Pseudocode 9 gives an overview in the training process of an ANT. The learning procedure is split into two parts: a *growth phase* and a *refinement phase*. For optimizing the parameters the negative log-likelihood (NLL) is used as the objective function to be minimised. Backpropagation [53] is used for the gradient computation whereafter gradient descent minimizes NLL for learning the parameters [18].

Growth phase
In the first phase of the training process an optimal architecture $\mathbb{T}$ is learned. The structure needs to be adequately complex given the training data. In the pseudocode, the while loop of line 4 to 14 is responsible for the growth phase. Starting from the root, a leaf node is taken in a breadth-first manner and changed by adding computational modules to it. One of three things can happen at a leaf (figure 3.16). The first is to split the leaf node and therefore the data by adding a new router. Initially, the identity function is chosen as the edge transformer towards each new solver. Secondly a deepening transformation can be added to increase the depth of an incoming edge by introducing a new transformer. As a third option the model can just be kept as it currently is. During the growth of the tree, each encountered node is locally optimised. The parameters of newly added modules (routers, transformers or solvers) are optimised by minimizing NLL while the rest of the computational graph is fixed. Both the processed model as well as the original are compared to each other using validation. If the new architecture improves the NLL, the algorithm continues with these added modules. If NLL is not further improved or even increases, then a roll-back to the previous model is made. We repeat this process to all new nodes until no more component additions improve on the validation test. The model improves in two ways: it deepens an edge seeking to learn richer representations or it splits the data which is equivalent to the soft partitioning of the feature space.

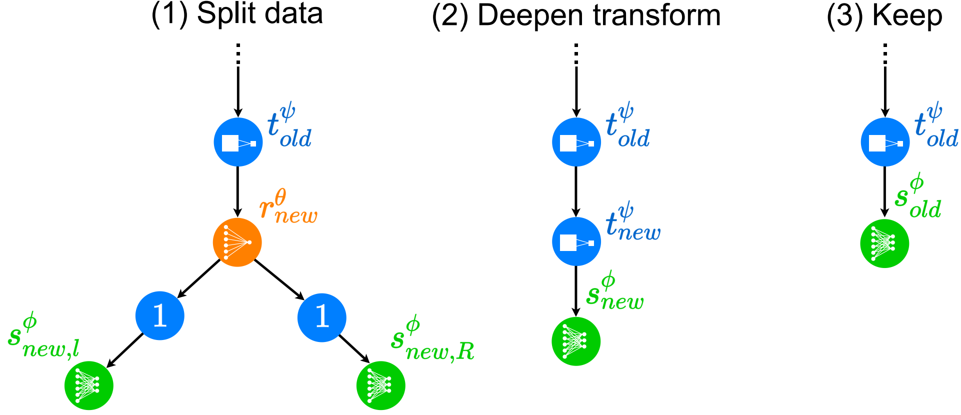


Figure 3.16: Possible actions the algorithm can take when in the growing phase. Either it can (1) add a new router with new solvers and identity functions on the edges to split the data or (2) add a new transformer node together with a new solver to deepen transform. A third option is to just keep the original transformer and solver and add nothing on that branch.

Refinement phase

The second phase is the one where global fine tuning of $\mathbb{O}$ happens. This is shown on line 15 of the algorithm. The architecture is now fixed and all parameters are again optimized with a gradient descent on NLL. This jointly optimises the hierarchical grouping of data to paths on the tree and down the associated expert neural network [18]. One possible option is to do global refinement during the growth phase. When applied, the user has to chose how many epochs of finetuning the algorithm would do after each addition of a node.

Algorithm 9 ANT Optimization

1: Initialise topology $\mathbb{T}$ and parameters $\mathbb{O}$ and one transformer
2: Optimise parameters in $\mathbb{O}$ via gradient descent on NLL
3: Set the root node *suboptimal*
4: **while** true **do**
5: Freeze all parameters $\mathbb{O}$
6: Pick next *suboptimal* leaf node $l \in \mathcal{N}_{leaf}$ in breadth-first order
7: Add router to l and train new parameters (1)
8: Add transformer to l and train new parameters (2)
9: Add (1) or (2) to $\mathbb{T}$ if validation error decreases, otherwise set l to *optimal*
10: Add any new modules to $\mathbb{O}$
11: **if** no *suboptimal* leafs remain **then**
12: Break
13: **end if**
14: **end while**
15: Unfreeze and train all parameters in $\mathbb{O}$

3.7 Explainable Reinforcement Learning

We gave in previous sections an overview of the current state of XAI. After closer inspection, we can observe that most XAI techniques are linked to supervised learning. By changing the feedback-giving component from the user to an environment, we could integrate XAI techniques into RL, creating *explainable reinforcement learning* (XRL). Especially in the case of DRL, where DL techniques make policies inherently uninterpretable, we can use the benefits from using explainability methods.

We examine a recent XRL taxonomy given by Heuillet et al. that encompasses several XAI techniques in a RL context [179] We draw parallels to our own taxonomy and add additional context from the few other survey papers available at the time.

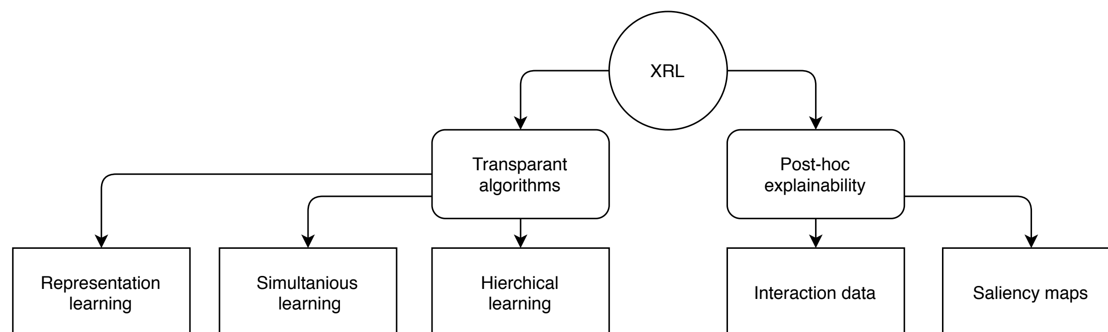


Figure 3.17: A proposed XRL taxonomy by Heuillet et al. [179].

3.7.1 Transparent algorithms

Transparent algorithms are similar to *inherently interpretable methods* or white boxes. Their strengths lie in the fact that they are designed to have a transparent architecture that makes them explainable by themselves, without the need of any external processing [179]. Examples of XRL techniques of this class are:

Representation learning

Representation learning algorithms focuses on learning abstract features that characterize the data [179]. This provides extra useful information about the features that have the advantage of having a low dimensionality, improving model performance. In a RL setting, learned features could be representations of state, actions or the policy itself, giving a better insight in the agent's behaviour. State Representation Learning (SRL) [180] is an example of a technique where the raw observational input the agent receives is used to model an observation space with meaningful states. This is use full in applications like robotics and other control problems.

Simultaneous learning

An explanation can simultaneously be learned with the policy. The learned explanation becomes an essential component of the model [179]. Methods involving *simultaneous learning* are useful on tasks where the introduction of certain knowledge is capable of adding insights into the learned policy. Examples are the classification of rewards by types and adding relations between states [179]. *Reward decomposition* [181] decomposes the reward function into a sum of reward types. This both results in an improvement in performance as well as readability. The related

decomposed reward DQN uses a vector-valued reward function to allow for decomposition. In it, each component is the reward for a certain type allowing for the comparison of actions in terms of trade-offs between the types [179].

Hierarchical learning

Methods that learn through hierarchical goals consist of both a high- and low-level agent [182]. The main goal is divided into sub-goals. A high-level agent tries to recognize these sub-goals. These are then given to a low-level agent whose task it is to achieve these goals in the environment. By learning what sub-goals are optimal for the low-level agent, the high-level agent forms a representation of the environment that is human interpretable [179].

3.7.2 Post-hoc explainability

Like we mentioned in previous section, post-hoc methods explain via an analysis done after the RL algorithm finishes its training and execution. Most post-hoc methods encountered in the overview were used in a perception context on visual input [179]. The work done by Coppens et al. is a form of post-hoc representation [14]. Same could be said about ANTs. Therefore, within this taxonomy, we could use the proposed KD technique in combination with SDTs and ANTs for performing XRL.

Interaction data

The agent behaviour can be explained by gathering data from its interaction with the environment while running, and analysing it in order to extract key information [179]. This technique is rather a secondary mechanism that produces the explanations for a black box.

Saliency maps

In image data, the most relevant elements could be highlighted to explain which parts are important to the agent. A saliency or heath map consists of a filter applied to an image that will highlight areas salient (a.k.a. noticeable) for the agent [179]. This makes saliency mapping a good technique to increase human interpretability. However, a disadvantage is the sensitivity to different input variations. This makes debugging not straightforward on the produced visual explanation.

3.7.3 Other literature

Puiutta and Veith made an overview survey based on the taxonomy of Adadi and Berrada [127, 128]. In this work, the main differentiators between certain techniques is the scope on which they operate as well as the time of information extraction. The scope could be local or global while the time could be during training (intrinsic) or afterwards (post-hoc). From there they concluded that all intrinsic methods are model-specific (which is straightforward) and that post-hoc methods could be both model-specific and agnostic. We also acknowledge the work done by Alharin et al. who based their taxonomy on the same distinctions as Puiutta and Veith did, but elaborated more on the types of input data including textual, images, rules and lists. Our work, as mentioned before, involves the usage of ANTs and SDTs to provide interpretability of learned DL models after the training process. This thesis, with the focus on ANTs as a viable surrogate model, is by categorization of the taxonomy a post-hoc technique of explaining learned DRL policies.

Chapter 4

Methods and setup

In this chapter we give an overview of the used methodology and setup we use for the experimental evaluation section. These are necessary to perform the research in the direction this thesis aims to go for.

The goal of this thesis is to test work done by Coppens et al. on different environments and to see if next to Frosst and Hinton's *Soft Decision Trees* and Tanno et al.'s *Adaptive Neural Trees* could be used as surrogate models for doing *knowledge distillation*. The main question is to test if ANTs are better models compared to SDTs in terms of performance and interpretability. We therefore have to both test performance by testing in a supervised/reinforcement learning context and test on interpretability by some measurement. We begin by talking about the proposed experimental setup we want to use in the actual evaluation. The experimental pipeline, consisting of three main phases, encompasses the training of the best possible DRL policy, the training of several candidate surrogate models and the evaluation of these models according to two criteria: interpretability and performance. Afterwards an overview is given of the used network architecture in all models as well as the main concepts and parameters behind the actor-critic model used in A2C and PPO. Before we start distilling, we need to train a DRL policy like PPO. We provide some implementation details as well as the used parameters for the different types of policies we trained. We give a short overview of all made adaptations to our used surrogate tree models. In the case of SDT this is mainly changes to the data loading mechanisms and representations of the outcome from the model. For ANT we introduce a method called SRDS (*smart routers, dumb solvers*) to improve the learning of router parameters during training to compensate for the powerful solvers. Because of out-performance by the networks in the solvers, the routers weights aren't optimized during training leading to noisy visualisations of their weight maps. By learning with sparse layers in the solvers we force the model to better optimize the parameters of the routers, improving readability on the long term. We discuss the problem of measuring complexity in a decision tree. Since no concrete measurement exists at the time of writing, we propose our own formulation of *visual complexity* that can be used to rank different architectures. We provide the idea behind our metric as well as its shortcomings and ways to improve it. Currently there are no available applications that can help providing insights in SDTs or ANTs and how their behaviour can be monitored in a XRL context. We developed a framework to help with the visualisation of trees as well as frame-by-frame analyses of the learned policies in their trained environment. Our developed prototype, called *Graybox*, is summarized in several points including its main motivations, technical details and interface capabilities. We give an overview of the functionalities of the system together with an explanation what it can provide while operated in a web environment.

4.1 Experimental setup

To have a standardized procedure for each of our experiments, we developed an experimental pipeline to train, distill and analyze policies. This guarantees that the best possible DRL policies (within the resources permitted to do this thesis) are chosen for generating the best datasets for knowledge distillation to be used to train tree models with good *explainability* and/or *performance* characteristics. The pipeline, shown in figure 4.1 is made of three main stages: *training and policy selection*, *knowledge distillation* and *analysis*.

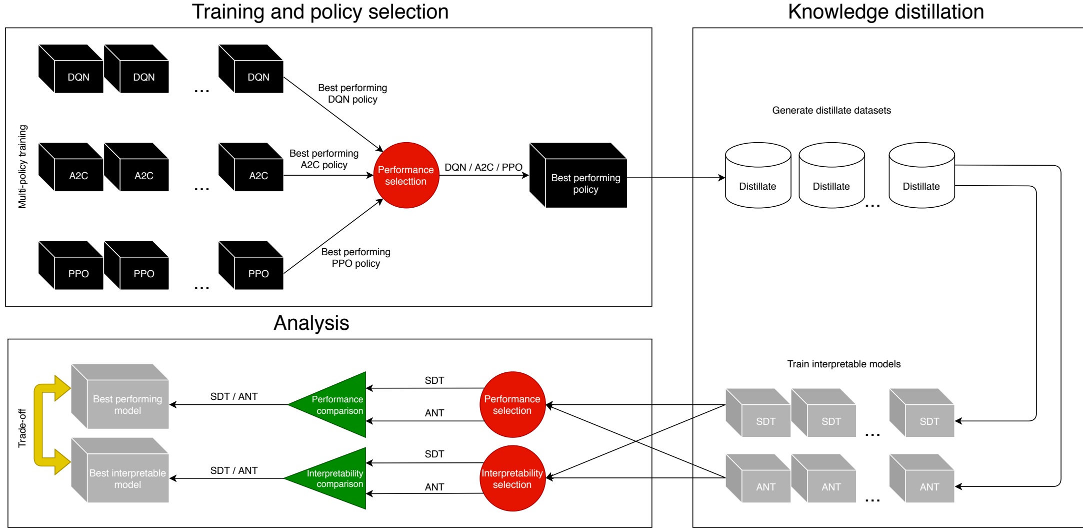


Figure 4.1: The experimental pipeline.

4.1.1 Training and policy selection

The first step in the pipeline is to train several black box models on a particular environment. For each algorithm, we train multiple DRL policies. By doing this *multi-policy* training, we increase the probability of obtaining a more optimal policy compared to others. There are several metrics for evaluating a RL policy including *regret* (the difference between a policy π and the optimal policy π^*) [184, 185], *cumulative reward* and *general intelligence test* [186, 187]. We opted to use cumulative reward over a number of episodes because of its straightforward interpretation in the case of gaming environments: the higher the score (e.g. reward), the better the agent plays.

From each type of learning algorithm we run an evaluation of 100 game sessions each to accumulate the total reward gained in those episodes. These are averaged over the number of game sessions and plotted in a boxplot for analyzing the performance of each algorithm. The best performing policy would be the one with the most total reward or average reward per game. This best performing policy is then used for the KD part of the pipeline, where a dataset will be generated with.

4.1.2 Knowledge distillation

The first part of the *knowledge distillation* phase is to generate a good dataset to train the surrogate models on. We let our chosen agent play a series of games where the amount of frames

is equal to a given value. This creates two datasets of distillates, one for all observations and one for all taken actions in the games. These are saved in separate files to minimize file size when eventually compressed. The accumulated score of games played during the generation is the indicator for deciding how good the dataset is. We generate several datasets and chose the best one based on that. Because we want to examine the effects of different set sizes, we opted to generate several datasets of different lengths.

The second part is training several neural trees of different types and with different structures/parameters. From each configuration, several models are trained to increase the sample size to have a better indication of the effects of each parameter combination. For SDTs we train a range of depths while for ANTs we try several combinations of finetuning and growing episodes. We do not change the hyperparameters of the models, which has an effect on the way they learn on the data. Further work can include this hyperparameter finetuning as part of the experimental pipeline to observe the effects they have on the learning process.

4.1.3 Analysis

The final phase of the experimental pipeline is the *analysis* stage where we compare all models according to several criteria. The first selector is to go over each type of SDT and ANT we've trained and select the most performant and the most interpretable one. The result is a structure per type of each of the two model criteria. Once we have our best performing ANT and SDT, we again select between these two the one scoring the highest on average on a series of games. For the determining the interpretability, we both use our complexity metric in combination with perceived complexity while trying to derive explanations from the tree visualisation. These visualisations are thoroughly examined on possible insights on the produced behaviours. Because these explanations lean more towards being subjective rather than objective observations, we try to convince the reader with gathered experiences from the tree structures in question. All of the made explanations are open for interpretation.

4.2 Network and policy architectures

In this section we briefly describe the chosen network architectures for our DRL algorithms together with a look on the used strategy for synchronising local and global networks in the actor-critic model.

4.2.1 The Deep Q-value network

Mnih et al. propose a network architecture in their original DQN paper inspired by the *ImageNet* architecture [188]. This was also the original motivation for using square frames with 2D convolutions as the GPU implementation of ImageNet required this. Several approaches to parameterizing the network could be considered, the most optimal having a separate output unit for each possible action with only having the state representation as the input [64]. In previous attempts [189, 190], the history of the agent together with their made actions were given as an input to the network, resulting in a cost that linearly grows with the number of actions. This is because then a forward pass through the network is required for each action that is given as an input.

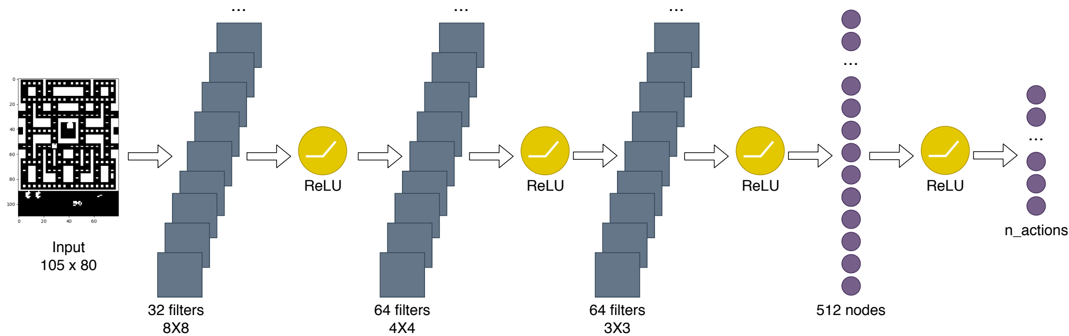


Figure 4.2: DQN network architecture.

The input frame, 105 by 80 pixels with one channel for luminosity, first passes through a layer that convolves 32 8x8 filters with stride 4 (being the number of pixels shifts over the input matrix when applying the convolution) [191]. A rectifier nonlinearity (ReLU) activation is applied afterwards [192]. The second layer convolves 64 filters of 4x4 with a stride of 2 followed by a convolutional layer of 64 filters 3x3 with a stride of 1. Both layers have a ReLU activation in between. The convolutional layers are followed by a final fully conected hidden layer of 512 rectifier units. The output layer is a fully-connected layer with output for each single action possible in the environment [38]. The architecture supports a number of actions between 4 and 18 depending on the game.

The chosen DQN implementation of *Stable Baselines 3* [193] uses the following parameters:

Table 4.1: Parameters of our black box DQN agent.

Name	Value	Description
batch_size	32	Input batch size for training.
learning_rate	0,0001	The learning rate.
buffer_size	1.000.000	Size of the replay buffer.
learning_starts	50.000	Steps before learning starts.
tau	1,0	Soft update coefficient (Polyak update).
gamma	0,99	Discount factor.
gradient_steps	1	Amount of gradient steps to do after each rollout.
exploration_fraction	0,1	Fraction of training period while the exploration rate is reduced.
exploration_initial_eps	1,0	Initial value of random action probability.
exploration_final_eps	0,05	Final value of random action probability.

4.2.2 The synchronous Actor-Critic model

A2C, a synchronous variant of A3C, uses a coordinator component to regulate the updates between the actors and the global parameters network. The algorithm waits for each actor to finish its experience segment before an update is performed [194]. This regularization results in a more effective use of GPUs working with large batch sizes [195]. We use the same feed forward network architecture as the DQN variant, but note the original paper also uses long short-term memory (LSTM) layers instead of linear ones. These are capable of holding memory of past activations. The implementation is also done in Stable Baselines 3 [193].

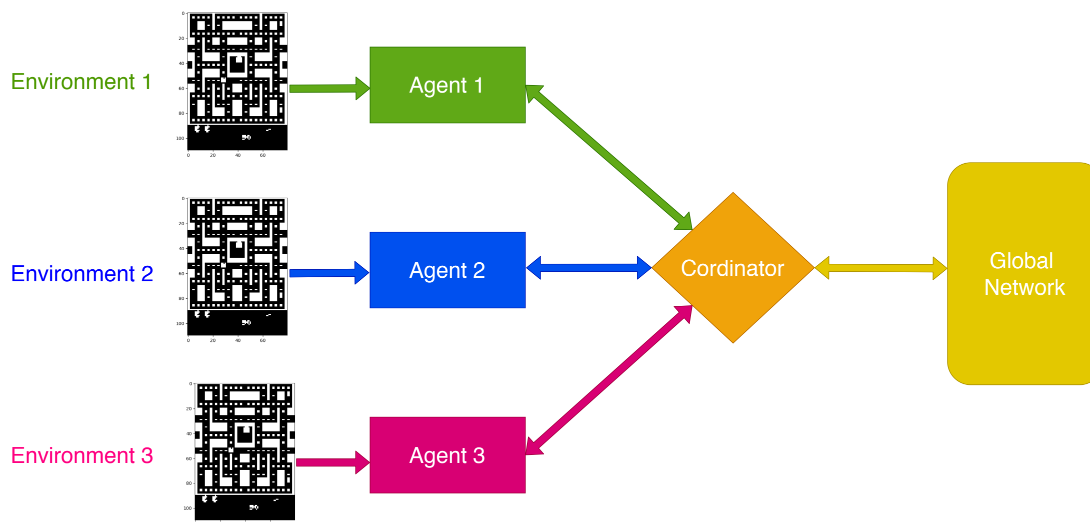


Figure 4.3: A2C network architecture.

The following parameters are used in the final DRL model:

Table 4.2: Parameters of our black box A2C agent.

Name	Value	Description
learning_rate	0,0007	The learning rate.
n_steps	5	Number of steprs to run for each environment per update.
gamma	0,99	Dicount factor.
gae_lambda	1	Bias vs variance trade-off factor for advantage.
ent_coef	0	Entropy coefficient for loss calculation.
vf_coef	0,5	Value function coefficient for loss calculation.
max_grad_norm	0,5	Maximim value for gradient clipping.
rms_prop_eps	1e-05	RMSProp epsilon.
use_rms_prop	True	Whether to use RMSprop (default) or Adam as optimizer.

4.2.3 PPO parameters

Finally, we briefly give an overview of the used parameters in the Stable Baselines 3 version of PPO, which uses the same CNN policy as DQN:

Table 4.3: Parameters of our black box PPO agent.

Name	Value	Description
learning_rate	0,0003	The learning rate.
n_steps	2048	Number of steprs to run for each environment per update.
batch_size	64	Input minibatch size for training.
n_epochs	10	Number of epoch when optimizing the surrogate loss.
gamma	0,99	Discount factor.
gae_lambda	0,95	Bias vs variance trade-off factor for advantage.
clip_range	0.2	Clipping parameter for the value function.
vf_coef	0.5	Value function coefficient for loss calculation.
max_grad_norm	0,5	Maximim value for gradient clipping.

4.3 Models and adaptations

In this section we discuss the changes made to the code base of the original ANTs and SDTs necessary to conduct the experiments of this thesis. The most important aspects of neural trees is that they combine hierarchical learning (what decision trees do) with deep learning (artificial neural networks). Both techniques are originally used in a supervised learning context but, through the use of *knowledge distillation* [17], can be used in a (explainable) reinforcement learning (XRL) context. For experiments with the MNIST dataset the code for both models stayed mostly the same. For the usage in XRL tasks situations, the original code of both models has to be adapted in order to work with a dataset of observation frames from the environments. For each model, we sum up the origins of the code base together with made changes:

SDT

The original SDT code used is made by Coppens et al. [1]. It is a PyTorch [2] [196] implementation based on Frosst and Hinton's SDT [175].

The data loading mechanisms of the SDT were optimised for handling MNIST data and such were designed to automatically download the dataset when needed. Because because we will distill policies from several black box agent, we will provide self-made data sets derived from these policies, implemented as a pickled Numpy array. We also added a validation set since the whole MNIST training set was used as training and validation was originally done with the test set.

An additional encoding of the targets needs to happen. Per array in the dataset representing the incoming observation, the corresponding action is saved as a single number. For it to work with the current implementation of SDT, it needs to be converted via a *one-hot-encoding* procedure to a binary vector of length equal to the number of actions available. A one in this vector at a certain index indicates the representing action equal to that index.

The original SDT code was made for doing batch training only and not prediction via inference. A *predict* method was added to do correct predictions with the tree. The method gets a batch of data as input and returns a tensor of the same batch length with the predicted targets. This allows for the addition of a method to calculate the accuracy of a given input batch.

ANT

The original ANT code is made by Tanno et al. [3]. ANTs use the same data loading mechanism as an SDT. The difference is where SDT uses a one-hot-encoding for the targets, ANTs only expects a single output value indicating the target class/action so no additional conversion is needed.

During initial testing of the experimental pipeline, we noticed noisy images from the routers with no resemblance of the environment in it. This indicates that the routers aren't properly trained during the learning process. Initially their weights are randomized and are over time optimized. The lack of training can be explained by the performance of the solver nodes which is much higher then the routers. We get relatively high accuracy in the begin when no routers are not present to the model. Afterwards, when they do get introduced, they don't contribute a lot to the performance of the tree. To solve this, we applied a new technique called *smart routers, dumb solvers* (SRDS). Instead of a fully connected linear layer for the solver during the growth phase, we only train on solvers with sparse linear layers decreasing their performance. This

[1] Source: `https://github.com/endymion64/SoftDecisionTree`
[2] Documentation: `https://pytorch.org/docs`
[3] Source: `https://github.com/rtanno21609/AdaptiveNeuralTrees`

allows the algorithm to better train the routers and disregard the solvers in the same node more. Afterwards, when the finetuning begins, the solvers are turned back to their fully connected form and the tree undergoes a global optimization with these layers included. SRDS uses a custom *sparse linear layer* implemented using PyTorch's sparse matrix to represent the weights. The amount of sparsity is set to low and depends on the environment trained. The first root solver should be able to classify just barely better than random while improvement would be visible with the addition of more and more routers.

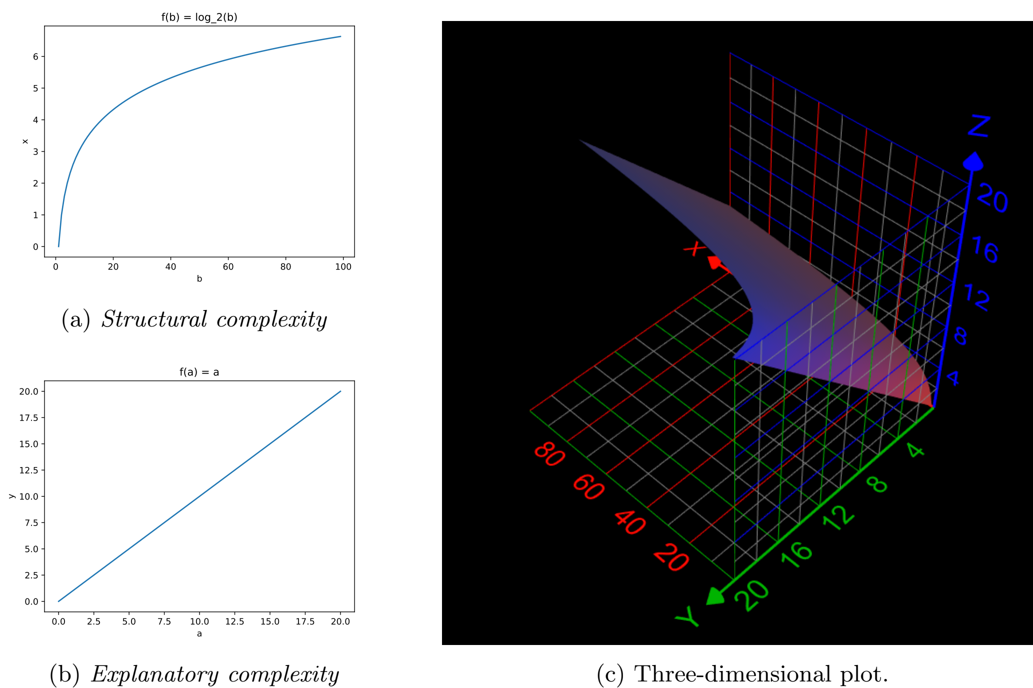
To better compare with SDTs, we opted to use the identity function as transformers for the ANT so no operations transform the input. As routers we chose to use a perceptron with size input size to 1 and as a solver we used a small neural network with as input size the amount of pixels on screen to the number of classes.

4.4 Measuring complexity

We want to assign a metric to the perceived complexity of a decision tree to measure its interpretability. It is a general rule that the more complex a model is the less interpretable it would be. It would be better to deliver simpler models if interpretability is required. While a single science of complexity does not exists [197], efforts of constructing a general theory of complexity are ongoing [198]. Previously made attempts tried to evaluate complexity by deriving from boolean functions [199] or rules [200] but these methods don't provide a comparative metric to analyse different topologies with resulting explanations of formed structures. Inspired by *computational complexity theory* [201], we define a complex interpretable system as a system that has a high computational complexity, denoted by the Bachmann-Landau notation $\mathcal{O}$. This means when a model is less computational complex, a human would interpret it easier. This kind of association has already been made in previous publications [202, 203, 204, 205], even with evidence that human visual perception of complexity outperforms that of a machine [206]. We propose a measurement for determining perceived *visual complexity* of a decision tree structure:

$$complexity_{tree,path \subset tree} = \Phi_k(tree, path) = log_k(nodes_{tree}) + nodes_{path} \tag{4.1}$$

with *tree* of arity k and *path* possible in the structure. This formula incorporates two possible views on the tree: a local *explanatory complexity*, as indicated by a *path* from root to leaf, and a global *structural complexity*, the whole composition of the model.



(a) *Structural complexity*

(b) *Explanatory complexity*

(c) Three-dimensional plot.

Figure 4.4: Visualisation of the proposed complexity formula $z = \log_2(x)+y$, with X representing the number of nodes in tree (*structural complexity*), Y the number of layers in the structure (*explanatory complexity*) and Z the complexity number Φ.

As mentioned by Molnar, models can be interpreted on both a local and global scale. For trees this is equivalent to an explanation produced by a *path* and the node interactions of the entire tree respectively. The *explanatory complexity* is given by the identity function. A linear path is computed in $\mathcal{O}(n)$ with n the number of visited nodes in the path thus a path's complexity grows linear with its length. Per full layer we add to a tree, the total number of nodes increases exponentially. For *structural complexity* we opted to chose the $\mathcal{O}(log(n))$ of a regular tree search [200]. This logarithmic function suppresses the exponential grow of nodes so the resulting value would grow linear with the amount of layers added. This formula is applicable for trees any arity. In the context of the used trees in this thesis, we define *binary visual complexity* as:

$$\Phi_2(tree, path) = log_2(nodes_{tree}) + nodes_{path} \tag{4.2}$$

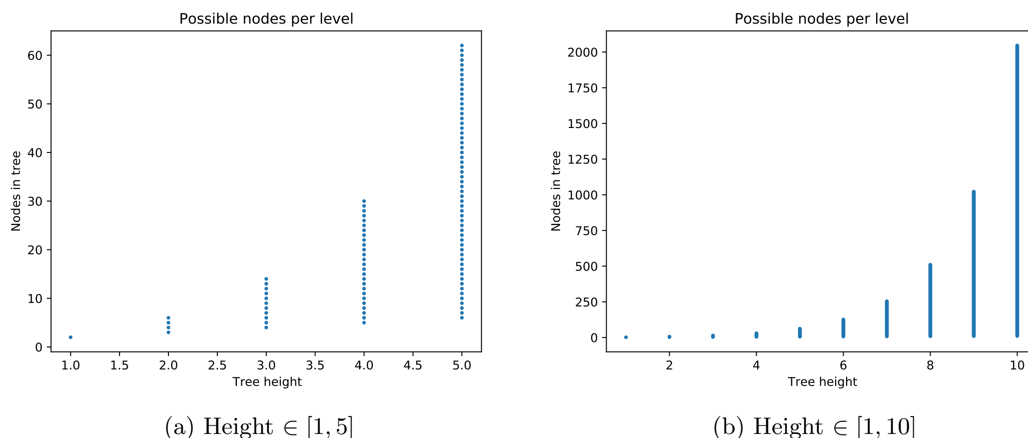
Three variants can be derived from Φ_2:

- $\mathbf{\Phi_2^{max}}$ using the longest possible path of the tree.
- $\mathbf{\Phi_2^{min}}$ using the shortest possible path of the tree.
- $\mathbf{\Phi_2^{avg}}$ the average length of a path in the tree.

Depending on the way complexity is defined (in a *best, worst* or *average* case), a different path selection criteria could be made resulting in one of the derivative formulas. Because the most common analysis is that of *worst case* big $\mathcal{O}$ notation, we go further with the Φ_2^{max} variant of the formula. A third derivative Φ_2^{expl} can be used to measure the complexity of a certain path *expl* formed in the tree.

We define a topology $t \in \mathbb{T}$ as a tuple (N^n, E_l^e) with N^n the collection of n nodes and E^e the set of e edges between the nodes $(n_i, n_j) \in E$ connecting the nodes and forming a tree of l levels high. Not all tuples produce valid trees. For instance, a binary tree with 4 nodes cannot have 2 or 5 layers. For l, the range of possible nodes is $n \in \left[l, l^2 - 1\right]$. The possible topologies are visualized in figure 4.5 up to a height of 10 layers.

It is a straightforward exercise to examine if the function is injective, meaning there exists no two topologies with the same complexity Φ_2^{max}. We can find two examples with a height of at most 5 in the domain for binary trees: $\Phi_2^{max}(5,7) = \Phi_2^{max}(4,14)$ and $\Phi_2^{max}(12,4) = \Phi_2^{max}(5,6)$. These examples are visualised in figure 4.6.



(a) Height $\in [1, 5]$

(b) Height $\in [1, 10]$

Figure 4.5: Possible topologies $\mathbb{T}$ for a binary tree.

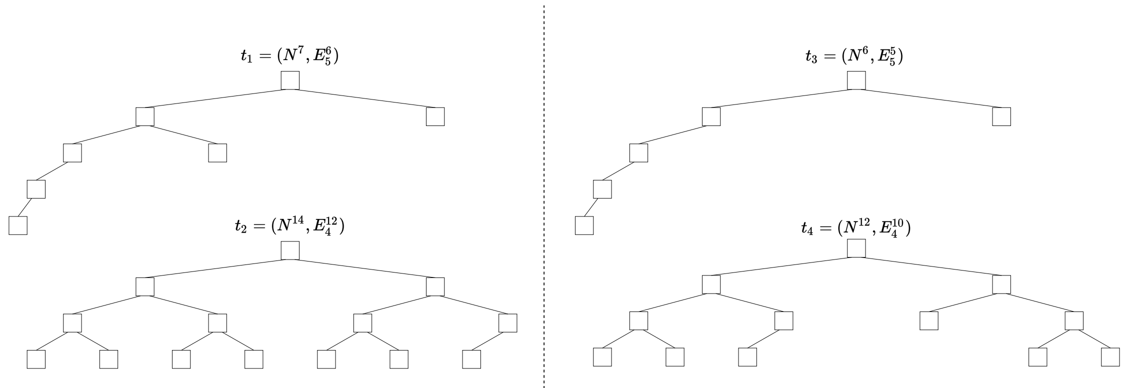


Figure 4.6: Binary trees with the same complexity value Φ_2^{max}. Left has value 7,8073549 while right is 7,5849625. The ones below are visually more complex than the ones above, purely from a objective point of view

We can notice a clear, intuitive difference between each pair of trees yet their complexity has the same value. In each column, the one above is less complex than at the bottom. In such cases, a better approach would be to calculate the complexity using minimal or average path. Opting for an average path results in a value dependent on the number of paths possible in the tree witch is on its turn dependent on the number of leafs. Φ_2^{avg} therefore incorporates a third variable into the formula

$$\Phi_2^{avg}(tree, path) = log_2(nodes_{tree}) + \frac{1}{n}\sum_{i=1}^{n} nodes_{path_i} \tag{4.3}$$

where n is the number of possible paths. This guarantees a more unique measurement per topology.

We note that it is possible multiple structures can be formed per topology node t. This has influence on the Φ_2^{min} since a minimal path could be of different length given different resulting structures. To counteract this, one could opt for the average minimal path of all possible trees. From table 4.4, which has the complexity measures of the different topologies, we could conclude that tree (5, 7) is less visual complex than (5, 6) if we use the average path metric. However, for the two other metrics (5, 6) would be least complex. In general, both structures could be intuitively classified as more interpretable compared to the other trees.

t	Φ_2^{max}	Φ_2^{min}	Φ_2^{avg}
(N^7, E_5^6)	7,8073549	4.8073549	**5.140688**
(N^{14}, E_4^{12})	7,8073549	7,8073549	7,8073549
(N^6, E_5^5)	**7,584962**	**4,584962**	6,0849625
(N^{12}, E_4^{10})	7,5849625	6,5849625	7,4182958

Table 4.4: Different complexity measures for the given topologies.

4.5 Prototype Framework

To facilitate the training of policies, generation of policy datasets, tree visualisation and distillation into simpler models, we developed a framework written in the programming language Python called *Graybox*. The philosophy behind the design of the application is summed in several points:

- **Easy to use** The user has a simple vocabulary of commands as well as a straightforward command line interface (CLI) to conduct experiments involving distillation from the beginning to the end. We restricted access to hyperparameters for finetuning the DRL policies and instead opted for using the ones proposed in their respective paper of origin.
- **Modular** Graybox consists of several modules that can be used as libraries for other code projects allowing the non-reliance on the provided interfaces. Dependencies on modules within the project are kept to a minimum.
- **Multi-profile** The storage of models, datasets and distillations is profile-based. This facilitates the used of the system by different users as well as conducting experiments with alterations to the code base.
- **Web integratable** Apart from the CLI, the framework provides a graphical user interface (GUI), executable in the browser. Its straightforward application programming interface (API) allows for the integration into other applications when desired.

We briefly provide the technical side of the application before introducing the reader to the capabilities of each of the two interfaces provided by the framework [1].

4.5.1 Technical details

The DRL training and playing is done via the `stable_baselines3` module [193], a *PyTorch*-based [196] variant of the original `stable_baselines` [207], which is a fork of OpenAI's `baselines` [194], originally implemented in *TensorFlow* [208]. The module allows for the training of models with only a few lines of code needed, greatly simplifying its integration into the framework.

The newly created visualisation module is based on the `ete3` library [209] for creating the global tree views. In it, nodes are visualized using the `cmap` method of `matplotlib.pyplot`[210].

Prepossessing of the OpenAI Gym [19] environment frames are processed using a custom wrapper based on the `WarpFrame` wrapper from `baselines` as well as processing via the `threshold` method of *Open CV* [211].

For the web server side of the framework we opted to use *Flask* [212] because of its lightweight Web Server Gateway Interface (WSGI) and flexibility. The API of the application can be found in the appendix.

4.5.2 Web interface

In this section we go over all the functionalities provided by the web GUI. As of the time of writing, only the *viewer* page of the application is completed with pages for training and distilling models in progress.

The interface for the *viewer* is split up into 2 main parts (figure 4.7). On the right side we have controls for loading a session on the page. On the left side the initial blank canvas would

[1] Source: `https://github.com/SenneDeproost/Gray_box`

be filled in by a view of the asked tree model when loaded. First a profile is chosen followed by the type of policy (original DRL model or distilled). One of the available models can be chosen from the profile and can be initiated with the corresponding algorithm and environment it has been trained upon. When *Load session* is pressed, the ALE will show up on the right side and when a distillate is loaded, the tree is shown as well on the left. The user can *step* one frame forward, *play* at a continuous rate or *pause* the session. When a session load is pressed again, the environment and agent will be reset from the start.

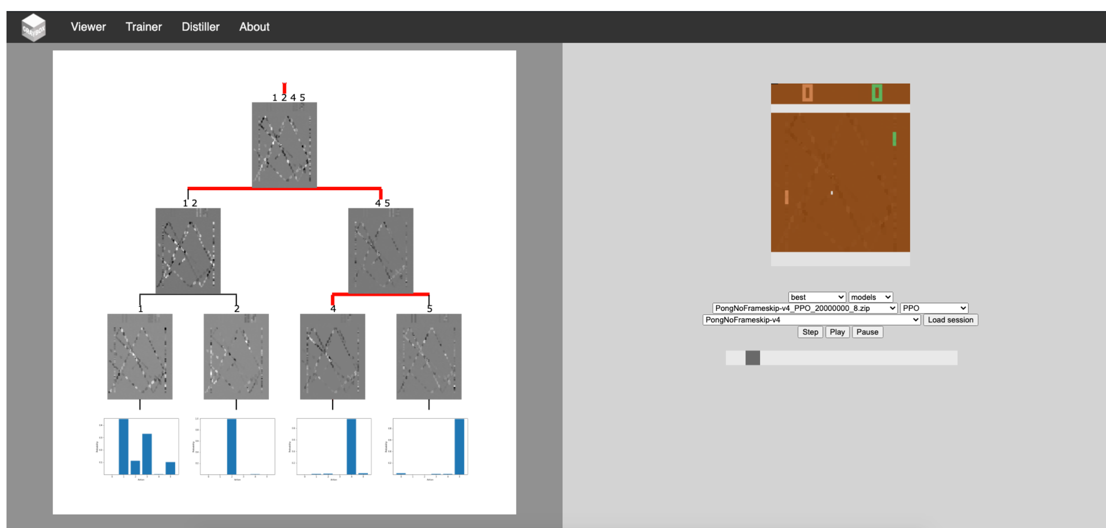


Figure 4.7: Overview of the Graybox tool.

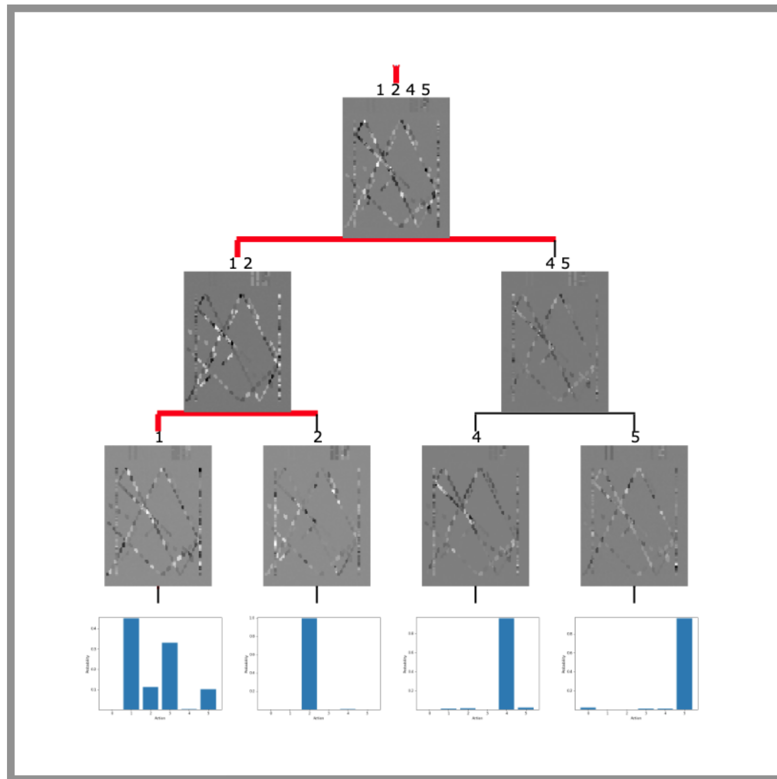
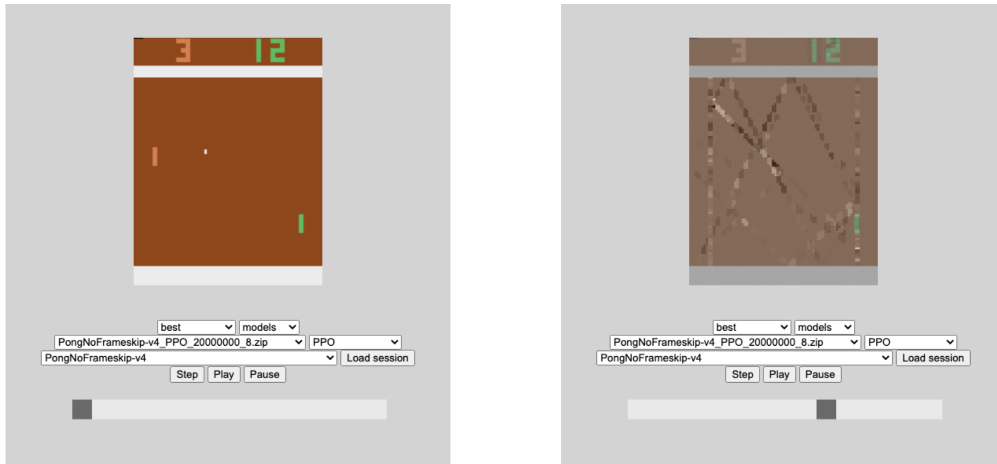


Figure 4.8: Global view of the tree model. Each node of the tree visualizes the weight map of that particular instance. The logits, representing the class distributions, are plotted in the leafs of the tree. The red path follows a trajectory towards the leftmost leaf node in the tree, which has the highest reaching probability.

The view of the game environment can be overlayed with the weights map of a certain node of the tree. By changing the slider underneath the control buttons, the opacity of this layer can be altered. To chose a node to be viewed the user can click on the respective node in the tree.



a: Gameplay only

b: With node overlay

Figure 4.9: View of the environment the agent interacts in as well as the controls needed to run and control the session.

The tree view indicates for each step of the agent the path to the leaf with the highest reaching probabilities with a thick red outline 4.8. When using greedy traversal, these would be the only nodes considered for a prediction. For a soft decision traversal, all possible paths of the tree are considered however. The plots below the leaf nodes indicate the values for the leaf logits, indicating which prediction is most likely to occur when arriving at that specific node. Numbers above the internal nodes indicate the classes of which the leafs have the highest logit values for. This roughly translates in an indication which part of the tree is specialized to recognize a certain class.

We use Graybox to examine the created DRL policies as well as their distilled tree derivatives. This is done for both subjective and objective measuring of the performed game play. Objectively we can analyze commonly used strategies by the agent which is more difficult to determine on numerical data alone. Subjectively we can compare the game play to that of a human and determine how *human-like* the agent plays.

Chapter 5

Evaluation

In order to be able to draw conclusion on the comparisons between our proposed models, we did an extensive analysis on several ML tasks. To begin, we give an overview of the used environment simulator and how the observations are preprocessed before training our DRL agent and tree policies on. This preprocessing is motivated with an example run and several considered techniques as well.

We did experiments on the classification of the well-known MNIST handwritten digit dataset, a commonly used benchmark. These allows us to test the capabilities of the neural trees and to examine their learning behaviour. This concludes the supervised learning (in a XAI context) part of our evaluation.

Within the context of XRL, we go through several passes of our experimental pipeline. We considered training several DRL algorithms (DQN, A2C and PPO) on five different Atari environments (Pong, Breakout, Space Invaders, Ms Pacman and Enduro). Learning these policies has been accomplished by using by using the computing infrastructure of both the VUB AI Lab cluster (Como PC 3) and the Hydra cluster at the VUB-ULB Shared ICT Services Centre. Results given in table 5.1, representing for each environment and used DRL algorithm the amount of trained policies. We chose for each environment the best performing policy in terms of accumulated reward during a test session and further went through the stages of *knowledge distillation* by using our created framework. The analysis part of each experiment examines the performance differences between our distilled models on both interpretability and in-game achievements.

Table 5.1: Number of trained DRL policies.

	DQN	**A2C**	**PPO**	
PongNoFrameskip-v4	9	10	10	**29**
BreakoutNoFrameskip-v4	4	9	10	**23**
SpaceInvaders-v0	4	5	10	**19**
MsPacman-v0	9	10	10	**29**
Enduro-v0	15	9	9	**33**
	41	**43**	**49**	**133**

5.1 Motivation and goal

The main reason for using structures like adaptive neural trees is their ability to learn their optimal architecture during training. This results in a (sub-)optimal structure that doesn't need post-optimizations like pruning. Soft decision trees from Frosst and Hinton have a static structure when trained and could lead to less optimal architectures. The main goal of this thesis is to repeat work done by Coppens et al. and see if adaptively growing trees can outperform statically ones in being a suitable surrogate model for knowledge distillation.

We included the supervised learning task of classifying handwritten digits as part of our evaluations because of the nature of RL tasks we want to learn. The training of a DRL policy can be translated into a classification task by learning for each given state what the best action to perform is [14]. For the actual KD part of the pipeline, where we generate a dataset of environment frames, we essentially let the model learn classifying its input onto the actions that would maximize its return. The quality of the dataset could be formulated as the amount of reward a certain DRL policy accumulated before a surrogate model is trained on it. This is the motivation behind creating and comparing different datasets in the pipeline.

For the actual RL part of the evaluation, our goal is to answer whether or not adaptive neural trees perform are more performant and/or interpretable by a user with limited domain knowledge. We only expect a basic understanding of how the game is played and which are the best strategies of a particular game. These are strategies such as using the power dots in Pacman until the last possible moment and the usage of the shields in Space Invaders to protect from enemy fire. Our comparison is done based on two criteria: which model is best in scoring the highest rewards in our games and which one gives the most insights into the internal decision making process of the model. As we discussed previously, the interpretability-performance tradeoff would be applicable for our resulting two structures as its consequences will be discussed.

5.2 Environment simulation

For learning real-world applications, for instance vacuum cleaning robots or self-driving cars, several approaches can be considered when formulating the environment. We could create a device capable of observing and interacting with the real world and couple the cognitive part of it into a software agent. Then, we can use the sensory data to observe the state of the world, passing down this information to the agent to decide upon an action to take and then perform the action immediately in the environment by actuating the device's output. Measuring the reward could be information directly given by an external observer or it could be judged based upon another piece of program programmed with desired outcomes in mind. The constraints of this immediate approach is bounded to time and can be slow. To accelerate the learning process, we could first model the real-life environment into a simulation. We could then deploy the agent in this simulated world and increase throughput by increasing the speed of which the simulation runs or by parallelising the process into several *workers* that gather high amounts of experiences that can be combined into one final policy. Once the policy is learned it could be deployed onto the actual device itself. Remark that perfectly simulating a real world environment is virtually impossible. We are bound to the capabilities of the machine the simulation runs on, which has a limit on the amount of detail we can represent the world. A balancing of computational resources between simulation and actual learning should be considered. It is not guaranteed that the learned policy when applied would not perform as well as indicated in the virtual world. Noise on both input and output is always possible and isn't straightforward to account for on a computer. A possible enhancement to this problem is the use of *domain randomization* where small randomized changes are made in the parameters of the simulation [23]. Because of these changes, enough variation is introduced to the training data of the learner so it could form a more generalized policy, decreasing the gap between virtual and material world [62].

5.2.1 OpenAI Gym

Gym is a benchmark toolkit developed by OpenAI that creates an integration between simulated environments and software agents [19]. It provides the user with a common interface providing episodic interactions for a diverse range of tasks.

The development of the module came from the need for good benchmarks that are easily integrated when comparing different algorithms. Existing benchmarks like the *Arcade Learning Environment* (ALE) [213] lack the convenience Gym provides which allows it to be more accessible for users from all kinds of background. The following design principles are considered:

Focus on environments instead of the agents interacting with it, which is up to the user to implement. This guarantees maximum convenience for the user and the freedom of choosing which style to integrate the agent interface with. Both *online learning*, where the agent updates its policy every step of the RL process, and *batch learning*, where observation and reward are collected separately to be processed later as one batched update, are supported.

Focus on sample complexity rather than final performance only. Benchmarking an RL algorithm can be done in both a performance-based as well as a time-based manner. The first one looks at the resulting accumulated reward to evaluate performance. This however can easily be increased when enlarging computational resources for the problem. A more comparable estimation of performance is measuring the amount of steps needed to gain a certain amount of reward. This threshold can be arbitrarily chosen but has to lie in a reasonable range of capabilities. This threshold is environment specific and could possibly be formulated as the maximum

reward possible in a task.

Clear indication of environment versions facilitating the reproduction of the task by other users. A change in version indicates a major difference in behaviour of the environment, even if it bears the same name as previously.

Inclusion of monitoring of the steps taken in the environment. This includes retrieving performance indicators as well as the ability to record video from a session.

Promoting peer evaluation instead of creating competitions. The OpenAI website has an dedicated section per environment for the comparison of performance of user submitted algorithms. Its main ideal is to share ideas and code to others by providing a performance overview as well as a useful benchmark for comparing the different submissions to the platform.

5.2.2 Preprocessing for DRL

Training on environments with high dimensional input data could be computational resource intensive and demand large amounts of memory. Even in the case of pixelated environments like Atari games the overhead could be significant. In the original DQN paper, Mnih et al. propose several preprocessing techniques to lower the input dimensionality of incoming environment observations [38]. To begin, they reduce the screen resolution from 210 by 160 to 84 by 84, resulting into an input array of 21.168 values instead of 100.800. Another improvement is to grayscale the images, only taking the luminance of all RGB (red, green and blue) channels combined. This gives us a total input of 7056 values, a 14-fold reduction from the original frames.

Another improvement besides reducing the dimensionality, is to use two or more consecutive frames in order to have a sense of motion and therefore time in the observation. However, because of the visualisations used in this paper, we would like to use only one frame as input in order to prevent a more complex observable input space for the user.

We propose a preprocessing method with the following characteristics:

- A reduced frame dimensionality with respect to the original screen ratio as the Atari environment.
- A more simplified color space to further reduce complexity and provide better compression when used in a dataset generation context.
- A reduced range of values a certain pixel can have.

In our neural trees we want to train, each pixel has an associated weight to it. Each weight value closely corresponds to its importance when activated by a certain input value. Because we want to visualize the weight maps formed in the nodes of our tree, keeping resemblance to the original environment input is important. That's why we opted to reduce the dimensionality of the Atari environment instead of squeezing it into a square. We chose to half the dimensions in pixels, producing frames of 105 by 80. This way, we can more easily overlay the weight map with the environment during visualisation. In figure 5.1 we show the different approaches to the Atari Ms Pacman environment.

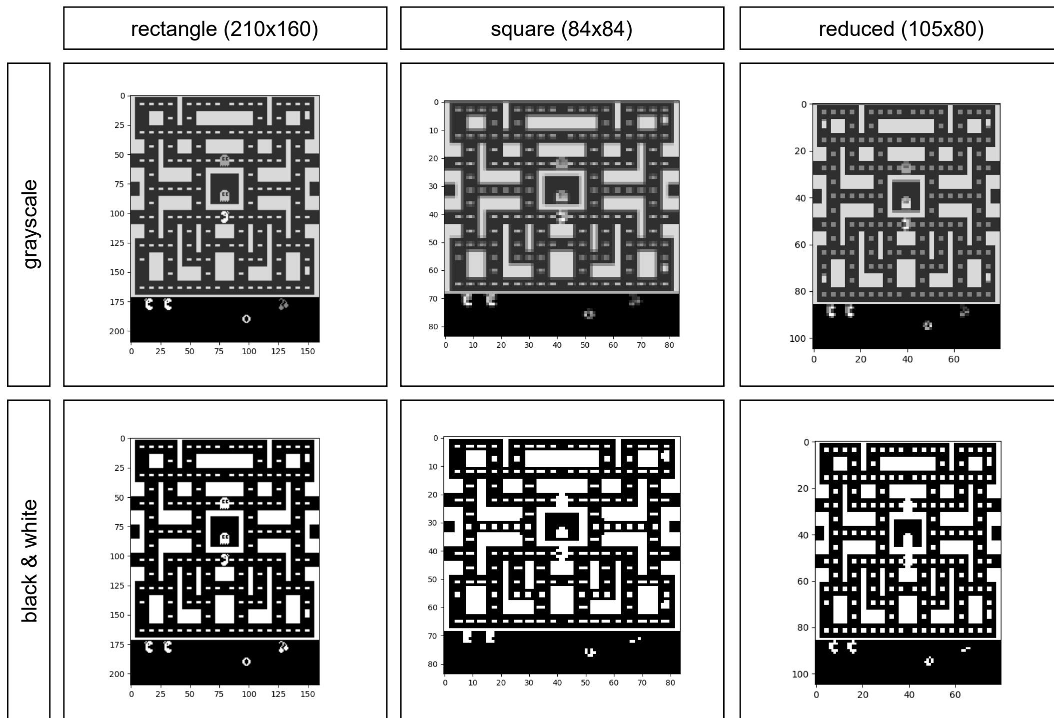
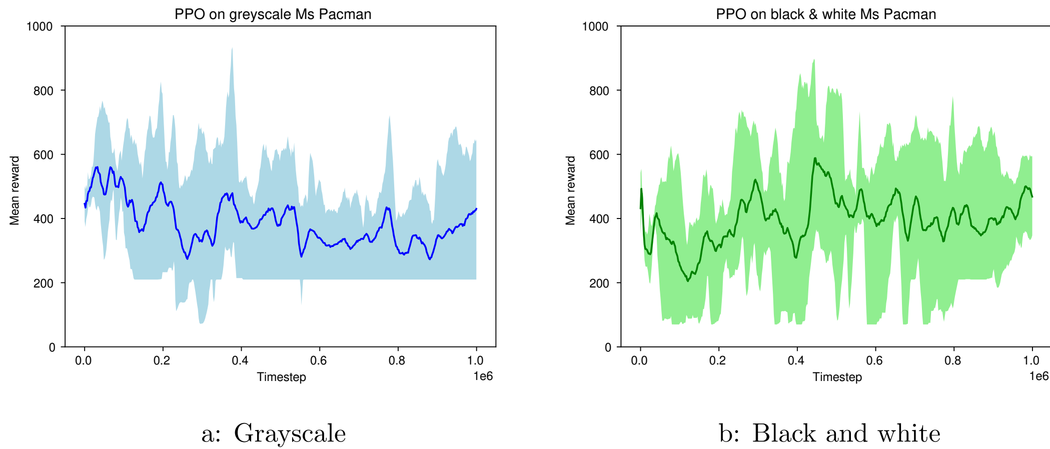
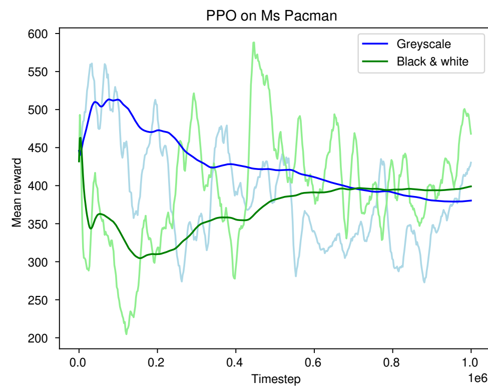


Figure 5.1: Comparing different frame dimensions for the Ms Pacman environment as well as different types of coloring. The ones on the left have the original resolution, while the middle ones are according to the DeepMind approach. Our reduction with respect to the original aspect ratio is shown on the right.



a: Grayscale

b: Black and white



c: Comparison between the two types of coloring

Figure 5.2: Mean reward curves for one million frames of Ms Pacman. Both (a) and (b) show the reward of playing five different PPO policies. (c) compares the two means the curves.

We conducted a preliminary test to see the effects of grayscaling the environment versus contrasting it to just black & white. The results are given in figure 5.2. We trained five different PPO policies per coloring of the Ms Pacman environment, each playing one million frames of the game.

When comparing both curves we initially see a high peak for the grayscale curve with a decrease towards the end. The opposite is true in the case of the black & white environment where we can observe a downwards trend to a bottom after which a rising trend emerges. From the comparison we could observe a better trend for in the black white case but with no large difference with the grayscale one. We note that we limited ourselves in this preliminary test to only the PPO algorithm, on one environment and only during one million frames. Despite this limited scope, from the observations made we could conclude that both coloring schemes produces similar results while training, if not better in the case of black & white.

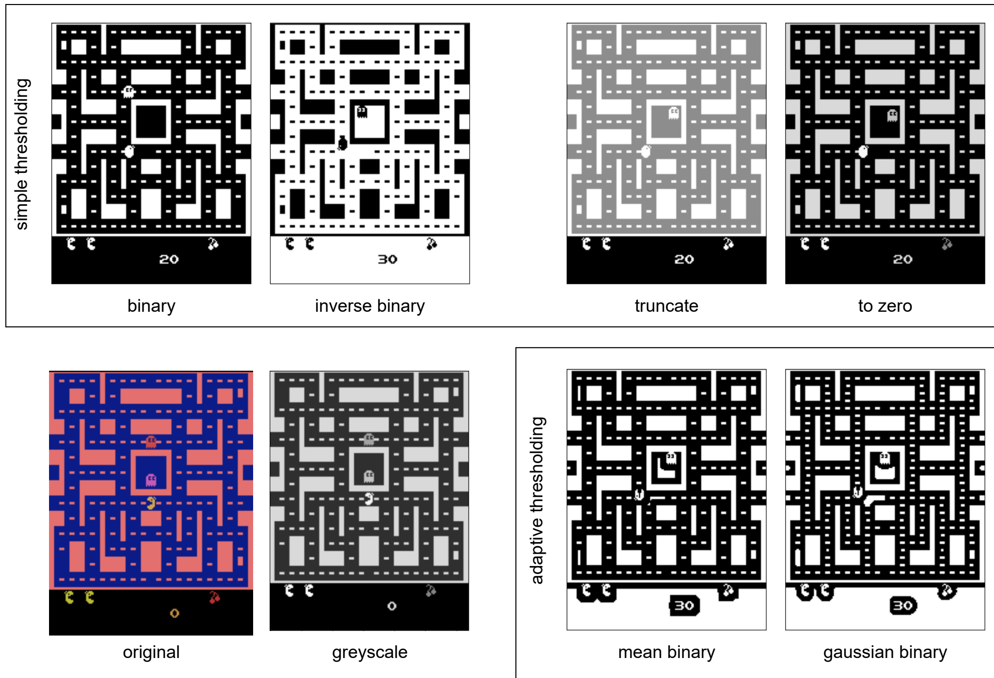


Figure 5.3: The Ms Pac-Man environment as processed by different thresholding methods. The above ones are simple one-valued results while below a tone measure is taken into account to determine the optimal threshold value.

We use the *WarpFrame* environment wrapper from OpenAI's *baselines* [194] to resize and grayscale the image. To convert the image to a black & white color space, we use OpenCV's *thresholding* method [211]. With this method, the same threshold value is applied for every pixel. This means if the pixel value is smaller than this value, it is set to 0. Otherwise it is set to a maximum value. In our case the minimum value represents black (0) while white is the maximum value (255). The result of using thresholding in the image is a matrix of either black or white values. To further simplify the frames, we decided to replace 255 by 1, leaving us with only binary data. This further improves the performances of our trees during training.

We examined both simple as well as adaptive types of thresholding. In the simple case, the *binary* method works as described previously while *inverse* changes the black and white values to 255 and 0 respectively. *Truncate* produces a more gradient change in tones while *to zero* does the same but only for the original values between 128 and 255. Between 0 and 128 the color is set to black. It is possible that a global threshold value is not good enough in some cases like when many colors are close to each other. Adaptive thresholding determines a local value based on a small region around the pixel. This generates a different theshold value for each region of the image, in theory providing better results in frames with varying illumination. We both can derive the value with the mean of with a Gaussian distribution making the areas "softer".

After considering each technique (figure 5.3) we concluded that adaptive thresholding generates to much unwanted artifacts in the frame. Icons have a black border around them and some game elements, like the dots in Ms Pacman, are jointed together. We chose to use the simple binary thresholding because of the simplified frames it produced. The inverted version is counter-intuitive in that the present of sprites like the ghosts are indicated in black while this color is most often associated with the absence of any element.

To chose a good threshold value for each environment we plan to use, we objectively observed the effect of different values on the first frames of a game. The processed image should visibly contain all game sprites important to the game. Figure 5.4 showcases different values considered. For instance, in the case of *Pong*, we have a usable frame when the threshold value is equal to 100. For *Enduro* this would be 50 because some lines of the road are not fully displayed at higher values. All threshold values are saved in a dictionary file for usage in the experimental framework.

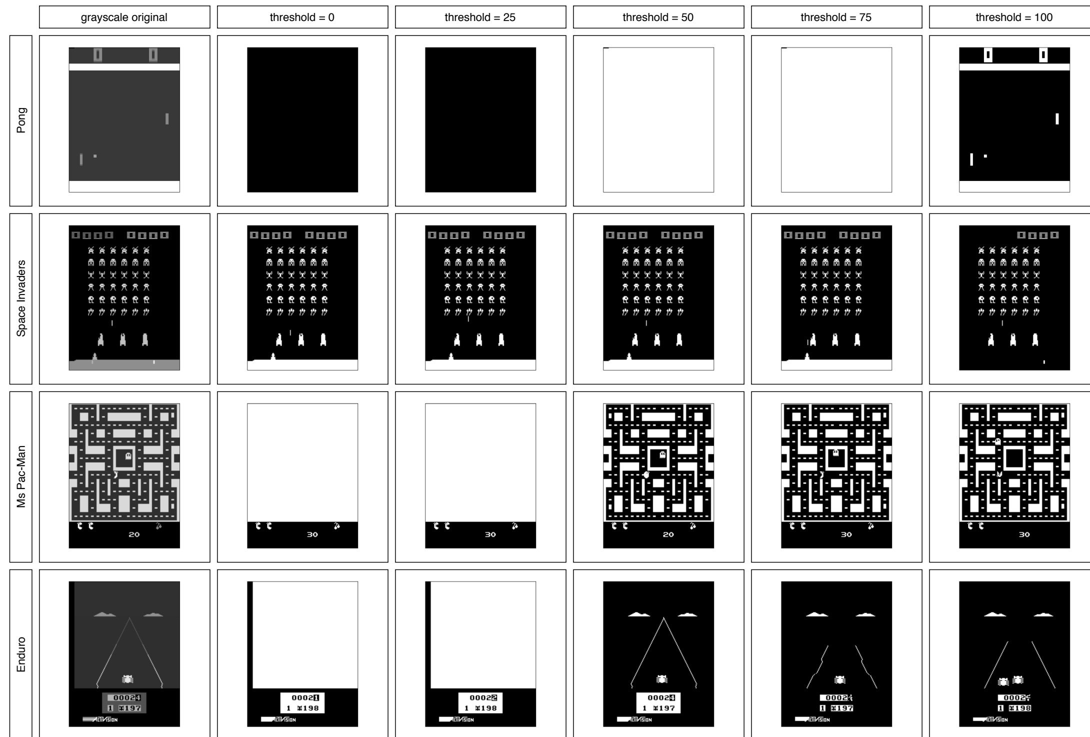


Figure 5.4: Different gaming environments with a series of varying thresholds. Each game has an optimal threshold for creating a binary color pallet which is dependent on the tones present in the original grayscale version.

5.2.2.1 Preprocessing for Neural Trees

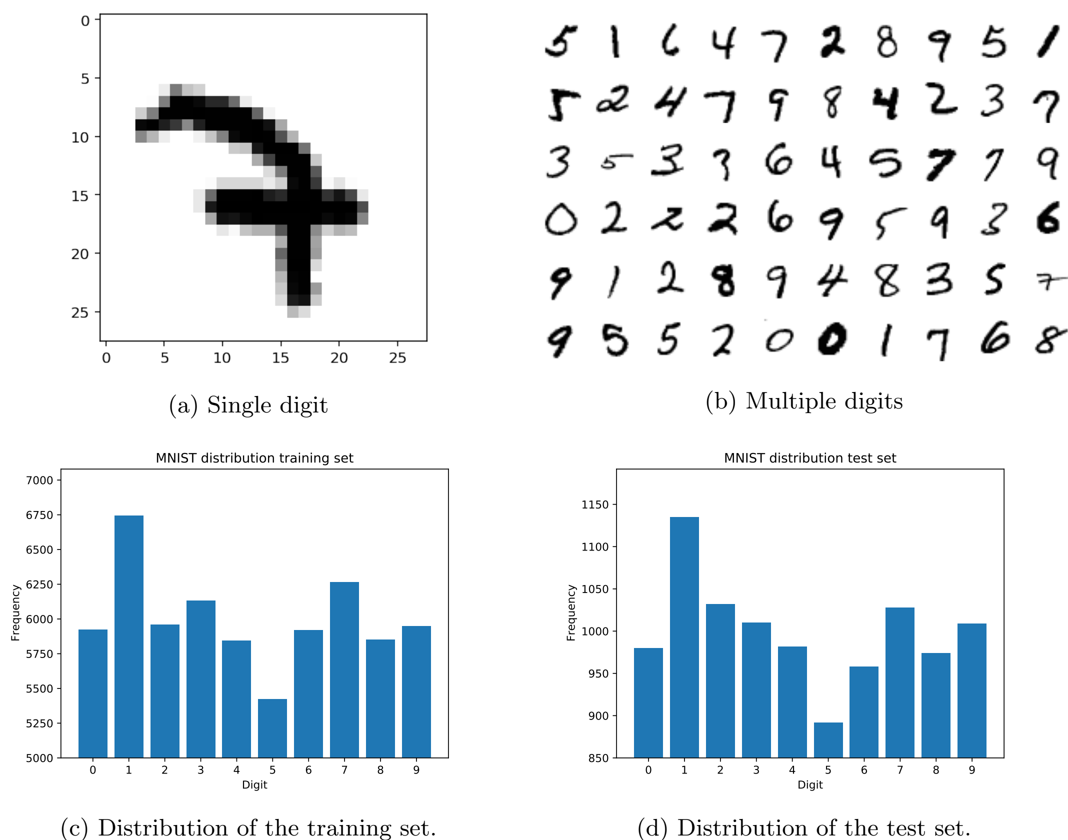
Because the nodes in our neural trees are simple one-layered perceptrons, they are limited to the amount of complexity they can learn. To further improve training and to lower our dataset file size, we decide to again lower our frame dimensionality by half. This results into frames of size 52 by 40 values, 2080 in total. We inspected the feasibility of this reduction in terms of resemblance to the original environment and concluded most most details important for the gameplay are preserved even at such low reduction. This also reduces the complexity of weight regions, simplifying the final visualisations when overlayed with the original frames.

5.3 Experiments

We first benchmark both SDT and ANT on a supervised learning task. This is the popular task of classifying handwritten digits from the MNIST dataset. Afterwards we train both types on tree algorithms in a RL context according to our pipeline. These are then benchmarked in the same OpenAI Gym Atari environments as they were trained in.

5.3.1 MNIST dataset

Our first experiments will be on the MNIST dataset [214] to compare performances between SDT and ANT for different parameters. This is a commonly used dataset to benchmark ML algorithms on. The *vanilla* variant of each model is used without any adaptations to the original code other then for retrieving metrics like accuracy, which was not implemented in the SDT code. The MNIST dataset consists of 60.000 training and 10.000 test examples of classified handwritten digits, as shown in figure 5.5. The inputs are 2 dimensional arrays of numbers representing greyscale values for each of the 784 pixels making up the input. The dimensions of the arrays are 28x28x1. As of writing this, the state-of-the-art model can predict the correct digit with an accuracy of 99,84% [215] while humans do it with 98% [216].



(a) Single digit

(b) Multiple digits

(c) Distribution of the training set.

(d) Distribution of the test set.

Figure 5.5: Samples of the MNIST dataset. Figure a is an example of a single digit with dimensions 28 by 28 pixels. Figure b is a random sample of multiple digits taken from the dataset. Figures c and d give the digit distributions of the training and test set respectively.

SDT

Our measurement Φ from section 4. is used as a numeric indicator for visual complexity. We use this metric to evaluate the complexity of the SDT models, as seen in table 5.2. Because every SDT is a complete binary tree, where the minimal, maximal and average path length are equal, the values of each indicator Φ are the same.

Depth	# nodes	$\Phi_2^{min} = \Phi_2^{max} = \Phi_2^{avg}$
0	1	**1**
1	3	2,5850
2	7	4,8074
3	15	6,9069
4	31	8,9541
5	63	10,9773
6	128	13
7	255	14,9944
8	511	16,9972
9	1023	18,9986

Table 5.2: Complexity of different SDTs.

We ran 5 training experiments, using different seeds for the weight initialization, for each depth ranging from 1 to 9. Results of these tests are given in table 5.4, with indication of the best values in bold. The predictions of these models were made using a deterministic selection in the leafs. The following parameters were used in every experiment:

Name	Value	Description
batch-size	64	Input batch size for training.
lambda	0.1	Penalty strength rate.
epochs	50	Number of epochs to train

Table 5.3: Fixed parameters during the MNIST experiments on SDTs.

In the columns table 5.4 we have several metric indicators for both accuracy and loss. *Min. acc.*, *max. acc.* and *avg. acc.* are minimal, maximal and average accuracy respectively. This is analogue for *min. loss*, *max. loss* and *avg. loss.* in the case of the resulting loss. Average is taken after 50 epochs of training and is the mean of all 5 trained experiments per depth instance.

Lowest average loss of 0,78692 is obtained by models of depth 5 whereas the loss rises from then on. This is because as the number of nodes grows exponentially, the loss would increase faster then it could be reduced by the gain in testing performance. So this does not mean SDTs tend to overfit on MNIST after reaching a depth of 5. Between depth 6 and depth 9 we observe an increase of only 1,976% in average accuracy for an eight fold in the number of nodes. Depending on user-defined criteria, one could be satisfied with a tree of depth 6 when complexity has to be as low as possible while retaining a good enough predictor. The confusion matrix in figure 5.6 shows the made classifications and misclassifications of the best performing SDT, with the numbers indicating the amount of predicted labels for a particular class. Most predictions happened correctly as can be seen on the diagonal. The classes with the highest percentage of incorrect predictions is for the number 3 with 51 out of 1026 cases (4,97%) of misclassification.

Test and training graphs from the best resulting experiments are given in figures 5.7 to 5.10. The transparent lines represent the actual data while the opaque ones are smoothed versions by a factor of 0,99. The same observations can be made: the deeper the tree depth the higher the gained accuracy on both the train and test set.

Table 5.4: Experimental data on trained SDTs of different depths.

Depth	Min. acc.	Max. acc.	Avg. acc.	Min. loss	Max. loss	Avg. loss
1	0	0,50	0,20684	1,872	2,183	1,93
2	0,0625	0,69	0,38488	1,434	1,809	1,4952
3	0,375	**1**	0,69438	0,933	1,637	1,05706
4	0,625	1	0,89002	**0,7169**	1,422	0,80826
5	0,6875	1	0,92558	0,7397	1,332	**0,78692**
6	0,75	1	0,94404	0,7969	1,321	0,8489
7	0,75	1	0,96056	0,843	**1,298**	0,90188
8	**0,8125**	1	0,959	0,9144	1,369	0,9582
9	0,6875	1	**0,9638**	0,971	1,423	1,0176

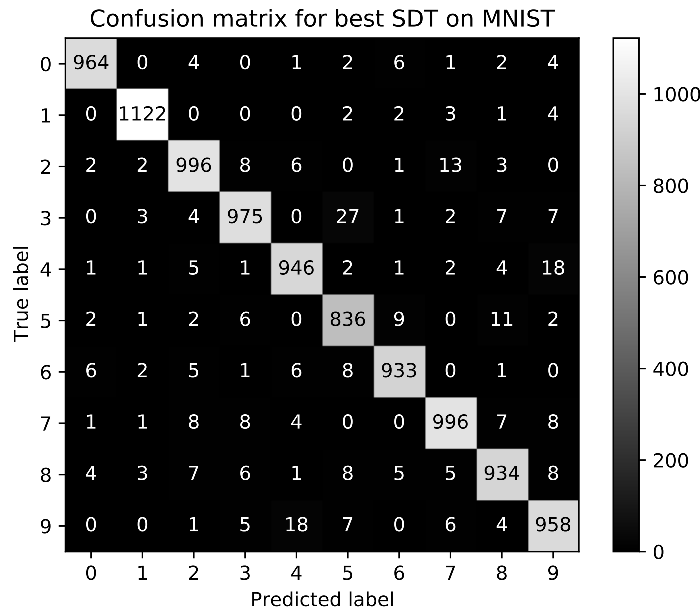


Figure 5.6: Confusion matrix of the classified digits with the most accurate SDT. We can see that the highest numbers are on one of the diagonals. These are the values where the algorithm predicted the correct label for a particular digit.

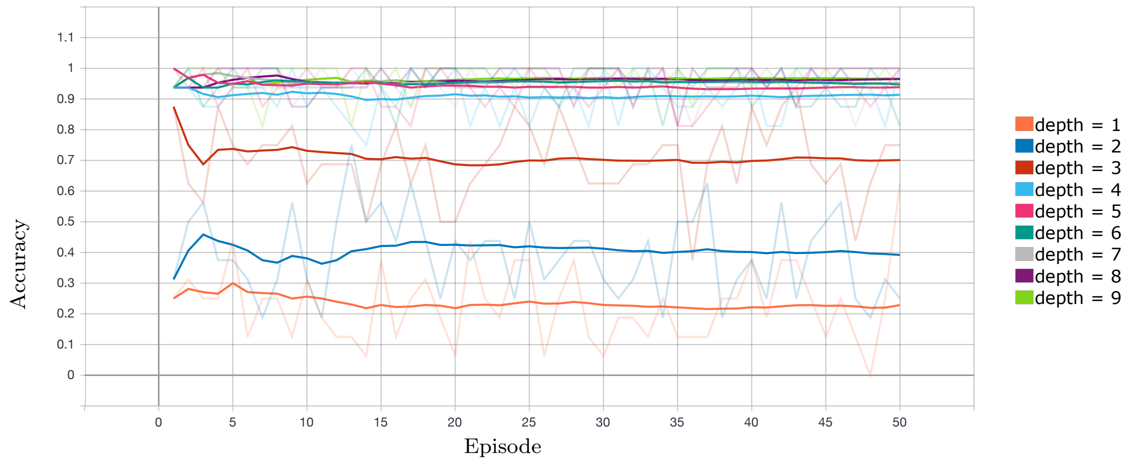


Figure 5.7: Best test accuracies of the trained SDTs on MNIST.

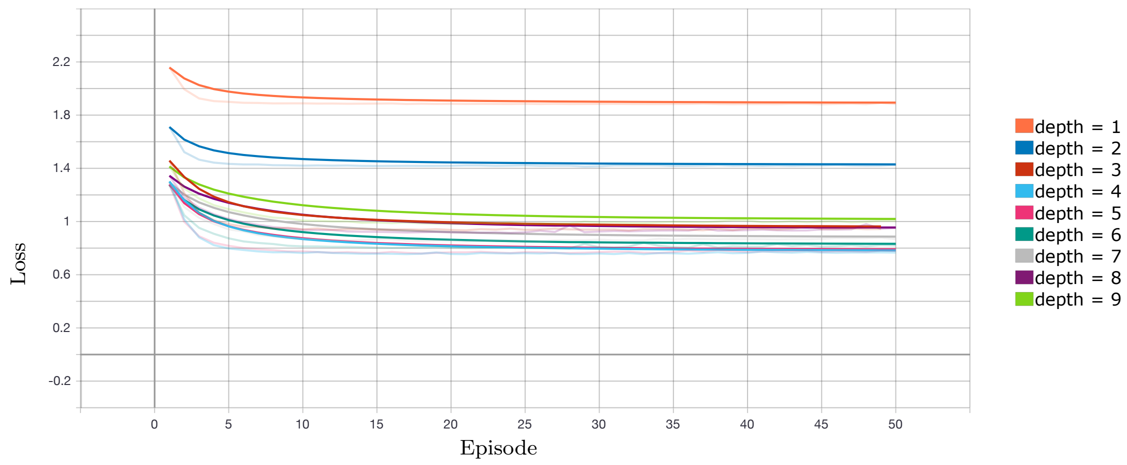


Figure 5.8: Best test losses of the trained SDTs on MNIST.

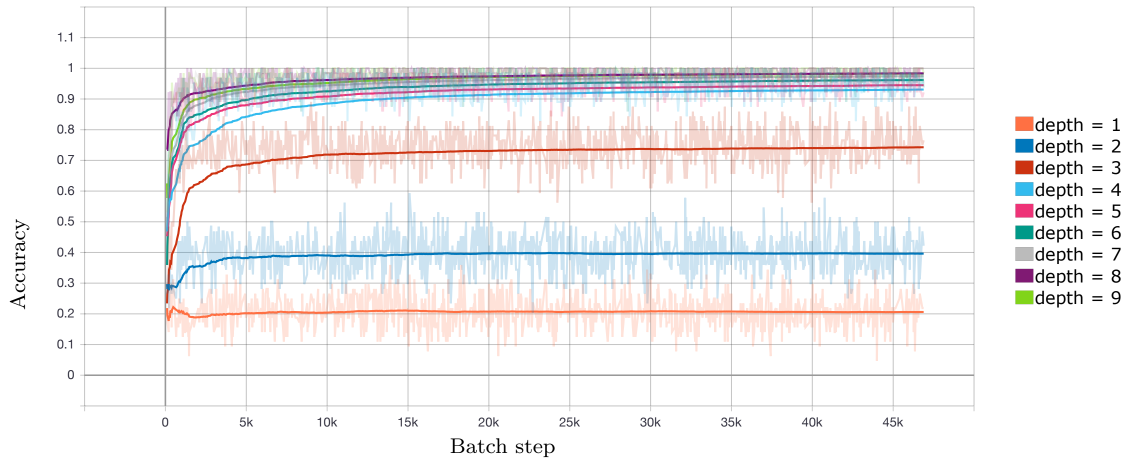


Figure 5.9: Best training accuracies of the trained SDTs on MNIST.

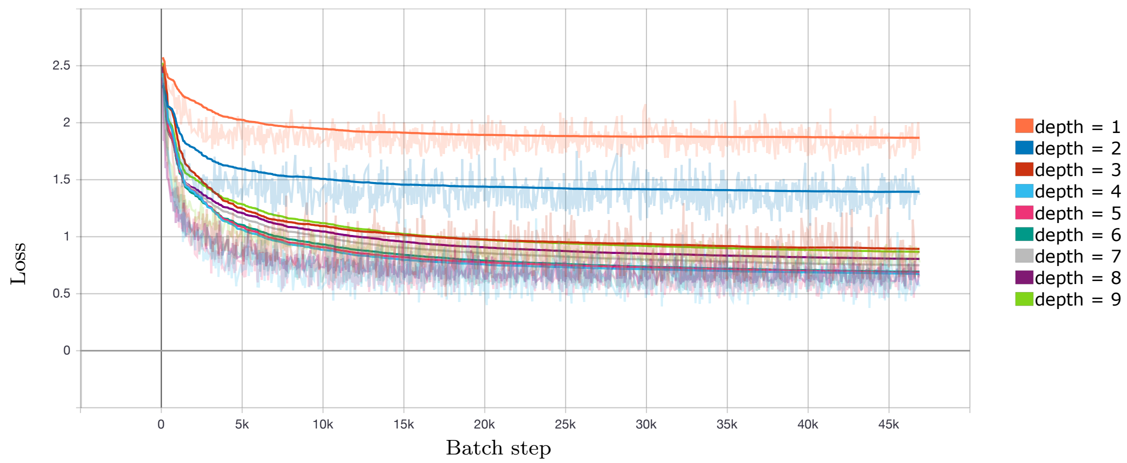


Figure 5.10: Best training losses of the trained SDTs on MNIST.

ANT

The main objective of MNIST benchmarking ANTs is to observe their formed structure as each model learns an optimal architecture during training. Transformations in the tree have been omitted in order to better compare with SDT models. The used transformers use the identity function to achieve this, by simply returning the input. We opted to use a fully connected (FC) linear layer 784x1 for the routing components and 784x10 linear FCs for the solvers. The maximum number of episodes dedicated for the *growth phase* at a node and the amount for the global *finetuning phase* are considered variables during these comparative tests. Finding the right amount of training in each phase is not straightforward depending on the learning task. The growth phase has a early-stopping mechanism with a patience variable when the tree tends to overfit for a number of episodes. The finetuning phase however has no regularization mechanism and could therefore more easily overfit when the number of dedicated episodes is too high. The following fixed parameters were used in all experiments:

Name	Value	Description
batch-size	64	Input batch size for training.
augmentation_on	False	Perform data augmentation (e.g. normalisation).
lr	0.001	Learning rate.
momentum	0.5	SGD momentum.
epochs_patience	5	Number of epochs to be waited without improvement at each node during the growth phase.

Table 5.5: Fixed parameters during the MNIST experiments on ANTs.

Breadth-first forward passing of the input is considered since this is the standard when training an ANT. To be comparable to a SDT, routers in the internal nodes decide upon soft decisions ensuring the same multi-path inferencing. Stochasticity is also introduced to the decisions of the routers. As with the case of SDTs, we trained five models per set of parameters. We varied the finetuning between 50 and 100 and chose values in the range of 5 to 50 for the growth phase, roughly doubling every time. The results are given in table 5.6. We can observe that the *finetune 100 / growth 20* model (abbreviated as F100/G20) obtained the highest average accuracy of 95,05% and lowest average loss of 0,1831. The highest accuracy and lowest lost observed are from a F100/G50 model with measurements of 96,15% and 0,136 respectively. If we look at the confusion matrix (figure 5.11) we again observe most of the made predictions to be correct. Noticeable is the misclassifications of the digit 8. In total, 1513 out of 7199 cases (21,02%) is misclassified.

To examine the effects of longer finetuning and/or growing, we can look at table 5.7 which sums up the structural differences between models.

Table 5.6: Experimental data on ANTs learned on different amounts for growing and finetuning steps.

Fine.	Grow.	Min. acc	Max. acc	Avg. acc	Min. loss	Max. loss	Avg. loss
50	5	0,9006	0,9567	0,94134	0,1576	0,4776	0,22084
50	10	0,7253	0,9606	0,93682	0,1399	1,088	0,23606
50	20	0,7916	0,9587	0,93304	0,146	0,7981	0,25162
50	50	0,899	0,9604	0,94486	0,1376	0,4096	0,20028
100	5	0,8626	0,9596	0,9463	0,1436	0,4303	0,2055
100	10	0,6531	0,9598	0,94916	0,1412	1,344	0,19032
100	20	**0,9129**	0,9584	**0,9505**	0,1422	**0,3105**	**0,1831**
100	50	0,9103	**0,9615**	0,94944	**0,136**	0,3186	0,1863

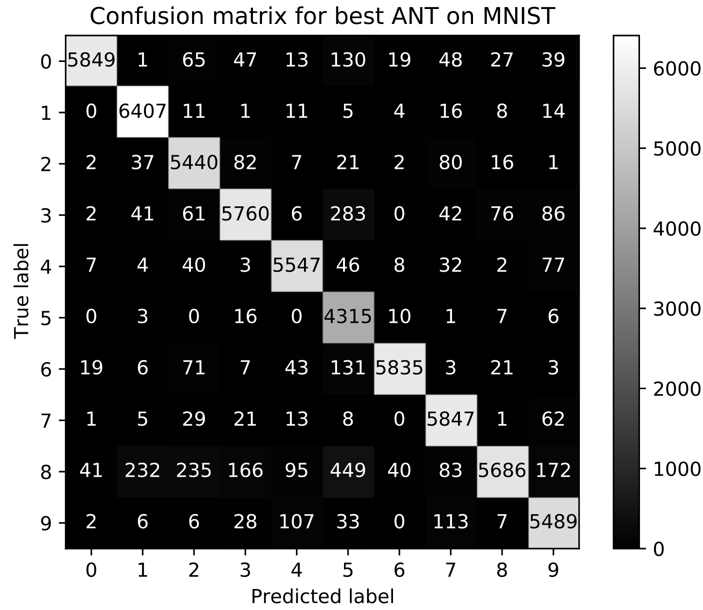


Figure 5.11: Confusion matrix of the classified digits with the most accurate ANT.

Table 5.7: Structural differences between the varying amounts of growth and finetuning steps.

Fine.	Grow.	Min. ep.	Max. ep.	Min. nodes	Max. nodes	Min. level.	Max. level.	Avg. nodes
50	5	**165**	**205**	**11**	**24**	**4**	**5**	**16,6**
50	10	337	651	16	43	5	6	29,6
50	20	360	823	19	31	5	6	26
50	50	364	871	11	44	4	6	25,5
100	5	225	395	12	39	4	6	25,5
100	10	387	689	21	51	5	6	29,8
100	20	489	857	21	46	5	6	31,2
100	50	321	896	12	41	4	6	30,6

For the F50 models we see a rise in average nodes from 16,6 to 29,6 followed by a drop to

25,5 when at F50/G50. This trend is different for the F100 experiments were average nodes tend to increase from 25,5 to 31,2 while slightly dropping to 30,6 at F100/G50. Height levels vary between 4 and 6 levels of height. These compositions from 11 to 24 nodes for a F50/G5 and 21 to 51 nodes for a F100/G10.

We note that the increase in average accuracy between the most and least accurate ANT is only 1,746% (table 5.6), which is considered rather small. In terms we only gain a small improvement for increasing the amount of learning time for growing and finetuning (resulting in more training episodes) and compared to the performance of the F50/G5 ANT an almost doubling in average nodes in the tree. If reduced complexity is traded in for this small percentage of accuracy, we could better opt for training an ANT with a lower number of F and G episodes. Complexity metrics are given in table 5.8.

Table 5.8: Complexity of different best performing ANTs.

Fine.	Grow.	# nodes	Min. path	Max. path	Avg. path	Φ_2^{min}	Φ_2^{max}	Φ_2^{avg}
50	5	**25**	**5**	**6**	**5,75**	**9,6439**	**10,6439**	**10,3939**
50	10	27	5	6	5,9231	9,7549	10,7549	10,6780
50	20	32	6	7	6,3571	11	12	11,3571
50	50	45	6	7	6,6190	11,4919	12,4919	12,1109
100	5	35	6	7	6,3125	11,1293	12,1293	11,4418
100	10	52	6	7	6,9166	11,7004	12,7004	12,6171
100	20	37	6	7	6,3333	11,2095	12,2095	11,5428
100	50	29	5	6	5,9286	9,8580	10,8580	10,7866

When comparing the structures of most accurate models, we can observe the least complex one is a F50/G5 model with 25 nodes and a average path of 5,75. This yields a Φ_2^{avg} of 10,39 which is also the lowest. The spikes in loss at the end of the tests as seen in figure 5.13 could be explained by the nature of the finetuning phase. All episodes of finetuning are executing without any early stopping mechanism, which could result in overfitting of the model. A solution could be the introduction of a patience component similar to the one used in the growth phase.

In conclusion: we can state that the more finetuning an ANT is given the higher average performance of the tree but with risks for overfitting the data. The longer the growth phase, the more nodes in the structure are added on average (although drops could occur compared to lower values) and the higher the accuracy of the model (for at least a small increase of about 2% within these tests).

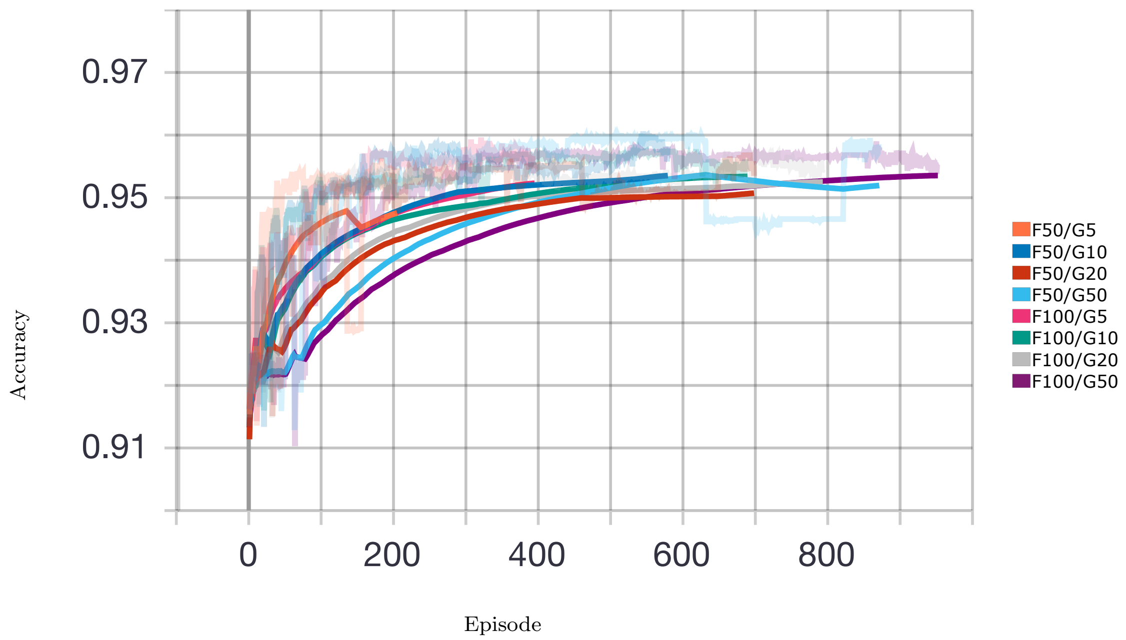


Figure 5.12: Best test accuracies of the trained ANTs on MNIST.

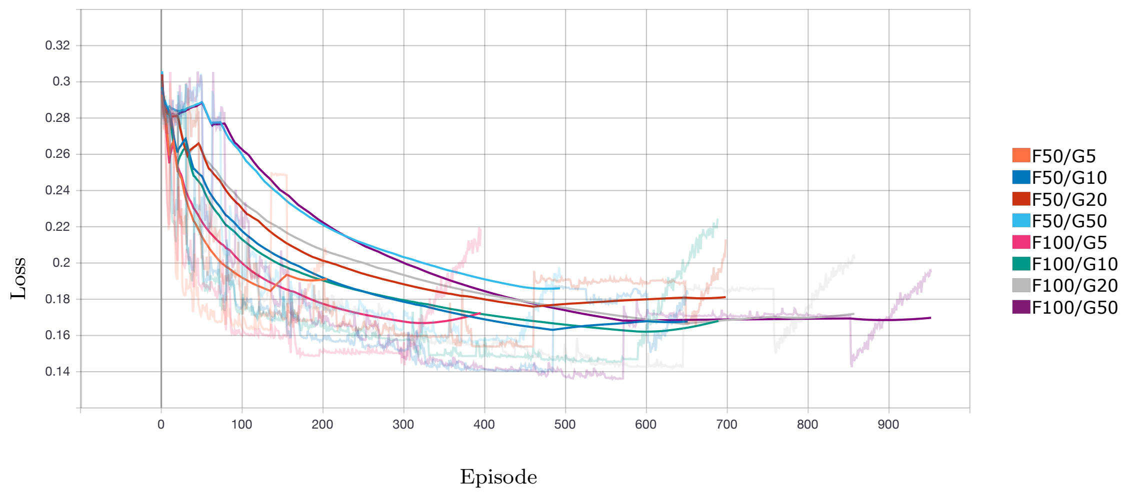


Figure 5.13: Best test losses of the trained ANTs on MNIST.

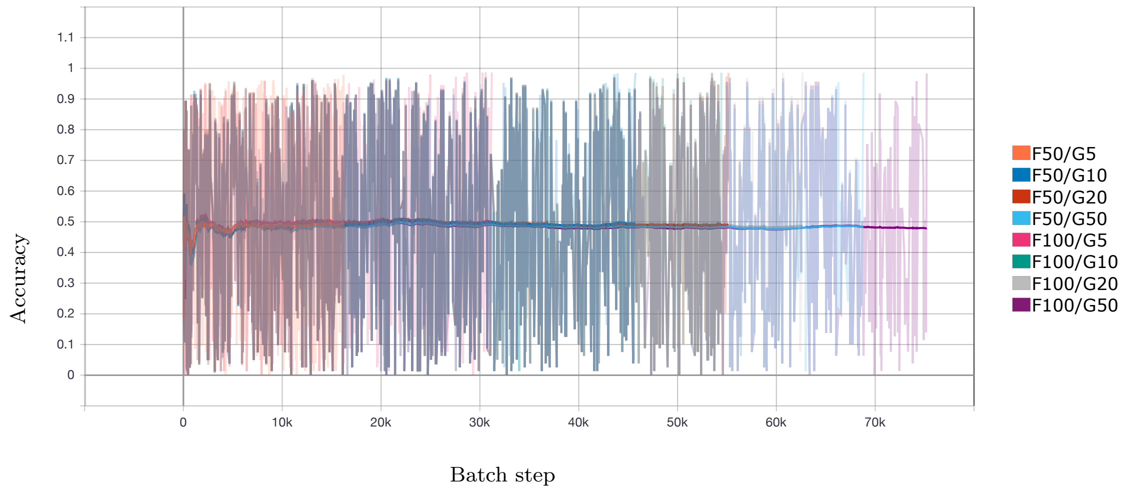


Figure 5.14: Best training accuracies of the trained ANTs on MNIST.

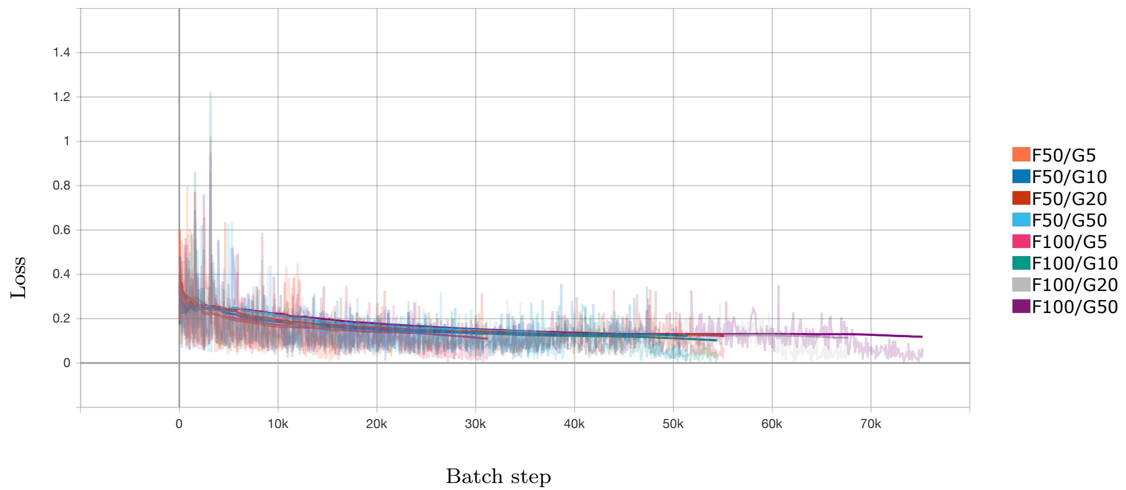
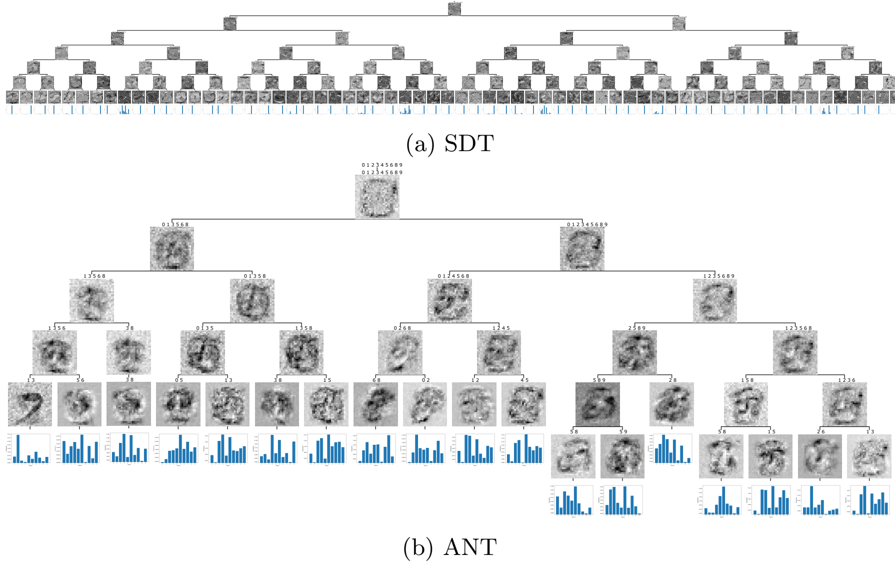


Figure 5.15: Best training losses of the trained ANTs on MNIST.

Comparison

We now have the necessary data to compare both SDTs and ANTs on the MNIST dataset. The most accurate SDT (depth 9) outperforms the most accurate ANT (F100/G20) by 1,33% on average accuracy. If we look at average loss, the best ANTs outperform the best SDTs by 0,60382. The latter however is a bad comparison since the difference in structures means the composition of the losses is different between models. To incorporate a structural evaluation, we performed experiments with the best ANT configuration (F100/G20, with an average accuracy of 95,05%) and trained SDTs with similar accuracy (depth 7, with an average accuracy of 94,848%).



(a) SDT

(b) ANT

Figure 5.16: An ANT model compared to a SDT with similar accuracy. (Larger views are present in the appendix).

Table 5.9: Comparison of both models on several metrics.

Model	# nodes	Avg. acc.	Avg. loss	Min. p.	Max. p.	Avg. p.	Φ_2^{min}	Φ_2^{max}	Φ_2^{avg}
SDT	255	0,94848	0,8979	7	7	7	14,9944	14,9944	14,9944
ANT	**37**	**0,9505**	**0,1831**	**5**	**6**	**5,3333**	**10,2095**	**11,2095**	**10,5428**

From figure 5.9 we can observe great differences in complexity due to the number of used nodes. While an SDT needs 255 nodes, an ANT learns an optimal structure of only 37 nodes to categorize MNIST with even greater accuracy. The different path lengths (minimum, maximum and average) are all 7 for an SDT while an ANT has shorter paths. This also translates into lower complexity metrics compared to ANT.

From our experiments we can conclude that SDTs are more accurate for greater chosen depths. If complexity of the tree is included in the evaluation, we can state that ANTs learn their structure more efficiently for a given accuracy on the MNIST dataset given enough training time in the form of finetuning and growing episodes. ANTs can be a more efficient model than SDTs when complexity per unit of performance needs to be as low as possible.

5.3.2 Ms Pacman

The *Ms Pacman* environment is a sequel to the popular *Pacman* arcade game from 1980. In it, you control Ms Pacman in a maze with *dots* scattered around. When a dot is eaten, the score is increased by 10. While moving, ghost enemies will try to hunt you down. On impact with Ms Pacman, the player loses a life. The player respawns at the center of the maze and when a third hit occurs the game is over. In the four corners of the maze the player can grab a *power dot* making Ms Pacman invulnerable and able to eat the ghosts who turn blue for a brief moment. A ghost can be eaten during that time, which rewards the player with 100, 200, 400 or 800 points depending on the number of enemies eaten in a row. Consuming the power dot itself is worth 50 points. At random time intervals, a bonus fruit will appear in the center of the maze. Collecting this bonus rewards the player a score between 100 and 500 depending on the type of fruit. The level ends when all dots are eaten. The level is reset, incremented and made more difficult for the player. The OpenAI Gym interface to interact with this environment consists of 9 discrete actions sequentially numbered from 0 to 8: 0 is the null operator doing nothing, (1, 2, 3, 4) are (up, right, left, down) and (5, 6, 7, 8) are (up-right, up-left, down-right, down-left).

Figure 5.17: The Ms Pacman environment.

Policy selection

As indicated by figure 5.18, the best performing policies are ones trained with the DQN algorithm. Out of 9 policies, 2 achieved average scores above 2000. None of the PPO or A2C policies performed above this. The best performing DQN policy achieved an average score of 2416. More than half of the trained DQN models achieve a score above 1500, with an average of 1447. The best performing PPO agent achieved a score of 1990, with all trained PPO policies having an average of 635 points. A2C only got a maximum of 775 and 226 points on average.

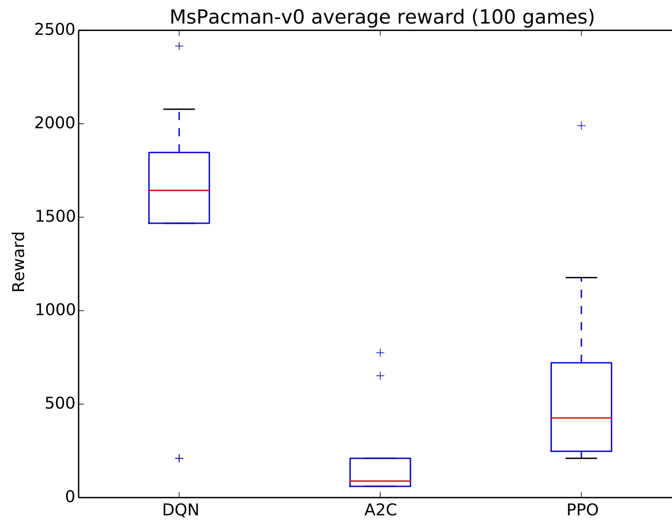


Figure 5.18: Boxplots of the trained Ms Pacman policies and their performance on 100 games. An outlier from DQN performed best with a score of 2416 points.

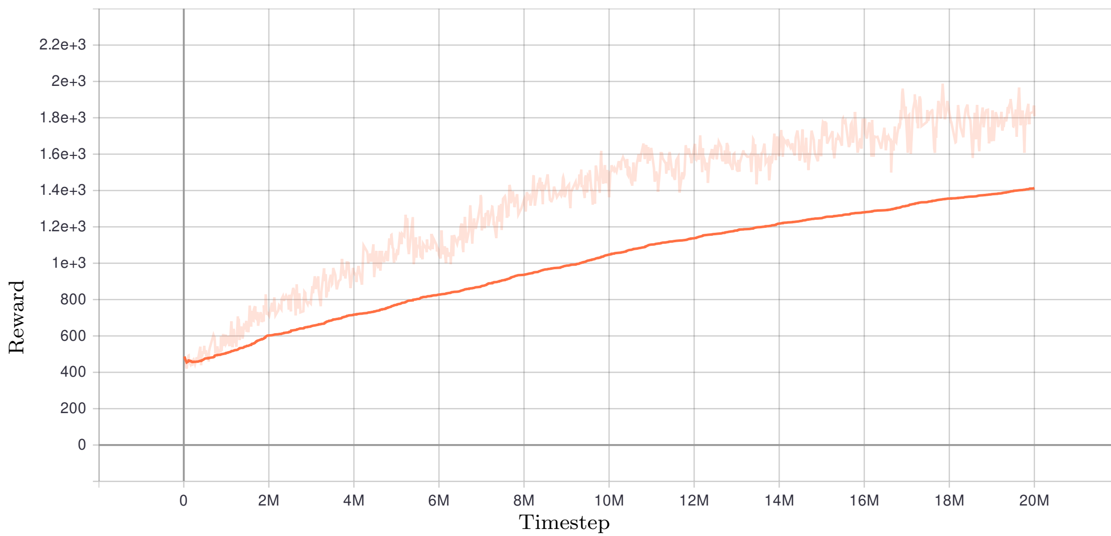


Figure 5.19: Mean reward curve of the best DQN agent. The transparant line represents the actual curve while the opaque line is a smoothed version of the reward (factor 0,99).

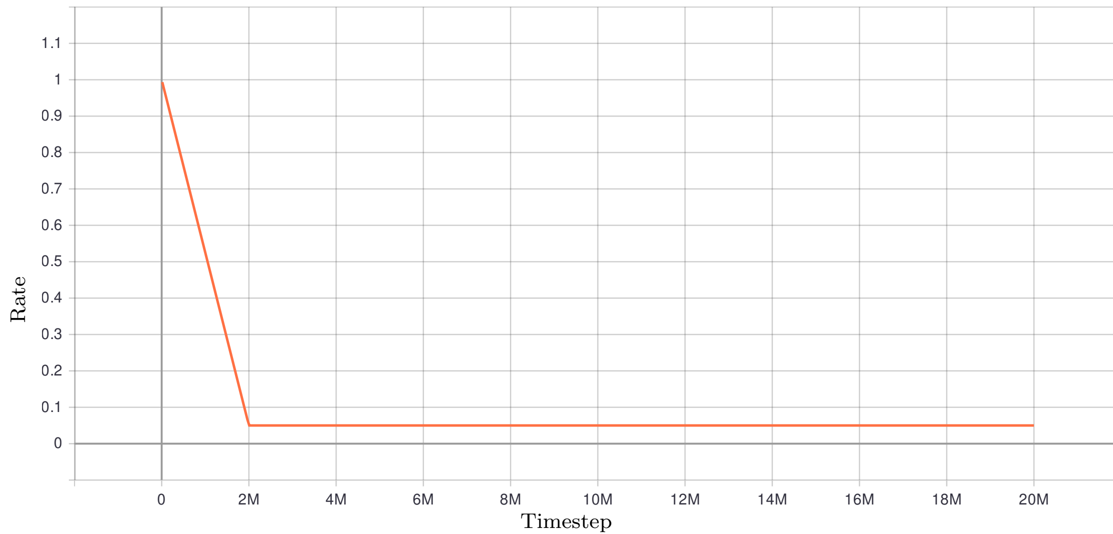


Figure 5.20: Exploration rate.

Knowledge distillation

With the best DQN agent, we created two datasets to perform the second step from our experimental pipeline: knowledge distillation. D_1 is a dataset containing 100.000 frames of the game played. This results in the recorded gameplay of 334 games with an average score of 2340 points per game. For a second generated dataset D_2 of 1.000.000 frames, 3467 finished games, the DQN agent played at an average score of 2335 points per game. A hypothesis we formulate is a bigger dataset D_2, therefore a larger explored observation space, would yield higher performance from the distilled policy. Whether this is true has to be concluded after testing.

Performance analysis

First, to determine the effects of different settings on ANTs, we examined a G20/F5 model to observe the effects of turning on and off stochasticity in the routers and enabling soft decisions or not. Enabling soft decision will result in a multi-path inference of the tree. This means that a decision is made by computing the predictive distribution as an average of the conditional distributions over all leaf nodes in the tree, weighted by their respective reaching probabilities. If soft decisions are disables, single-path inference is performed where hard decisions based on one solver are made. Stochasticity determines the behaviour of the routers. Stochastic inference (performing probabilistic routing based on the probability of going left or right) is used when traversing downwards with stochasticity enabled. If not, a greedy traversal is made using the routes with the highest probability.

We want to test these different settings of traversal in order to obtain the optimal parameter configuration that, given a certain distilled policy, yields the highest scores. To consider these options, we set up four copies of an identical tree but with different options enabled. This tree was trained on the smaller D_1 dataset with stochasticity and soft decisions enabled. To examine the performance, we calculated the average of 5 sessions of 100 games for each of the option combinations.

Table 5.10: Effects on performance of different settings for the router components of ANTs on the MsPacman environment.

	stochastic	**deterministic**
soft decision	809,38	**833,92**
hard decision	759,50	776,53

From table 5.10 we can observe maximal performance of 833,92 points when the routers are deciding upon soft decisions while being deterministic in the routing they provide. The lowest performance of 759,50 is gained when hard decisions are made using a stochastic routing function. With this information we conduct our experiments with option *stochastic* to be false as well as true for *soft decision* for the ANT models.

For ANT models, we opted to train F5/G5 models (5 finetune episodes, 5 growth) and F50/G20 models (50 finetune, 20 growth) on both datasets D_1 and D_2. Results are plotted in scatterplot 5.21 where the gained average score is compared to the amount of nodes in a tree. The drawn regression lines approximate the trends of the data points of using different datasets. We can observe a best performing model of type F5/G5, trained on D_1, with a gain of 1115,94 points on average. The lowest performance is 634,94 while the average of all trained models is 865,46. We see no correlation between models that have trained longer (e.g. F50/G20) and the performance of the model. Also the tree complexity in terms of nodes is not related to the amount of training. The larger size of a dataset could be the reason for larger trees since on average a tree trained on D_2 has on average 9,1 nodes while one trained on D_1 has 6,6. For each regression line the slope R is indicated. R_{D_2}, the slope for D_2, is approximately -6,32. This means for each extra node in the tree, a model trained on D_2 decreases its average performance with -6,32 points. A more pronounced negative trend is visible for R_{D_1} with a value of ≈ -56. The slope of all models combined is given by $R_{D_1+D_2}$, which is also negative. From the made sample observations we can conclude that for both datasets the performance of a model decreases with the number of nodes present in the tree. A conclusion on the relation between training time and performance cannot be made since longer trained models do not result in better performances.

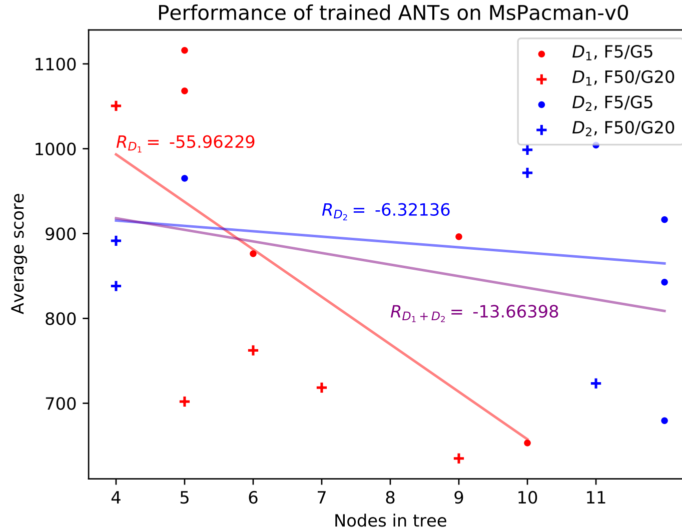


Figure 5.21: ANT models and their performance compared to their node complexity. Performance is the average of 5 sessions of 100 games in a row with random seeds. From this, we can conclude that the bigger dataset D_2 produced trees with more nodes and D_1 the best performing one with a F5/G5 model. The relation between performance of a tree and the amount of nodes is a negative trend for both datasets.

Before deciding upon the best performing model in terms of score maximisation, we first examine if the produced score distributions of playing 1000 games can be fit with a normal distribution. We first examine the case for the ANT and afterwards potentially the SDT.

As null hypothesis H_0, we state that the average score distribution of the best performing ANT model is normal distributed. For H_0 to be acceptable with a confidence of 99,5%, we decide upon the value of our threshold α to be equal to 0,05. Figure 5.22 visually hints at a Gaussian distribution because of the similarities between the histogram and the plotted normal curve. However, if we plot using a *Quantile-Quantile* (QQ) plot, we observe a rather significant deviation of the data from the line that indicates normality (figure 5.23).

To statistically test on normality, we first test with *Shapiro-Wilk*. The resulting p-value, using *SciPy*, is 8.76927127869351x10^{-24}, which is lower than the α threshold for H_0. This means we reject H_0 based on the *Shapiro-Wilk* test and with the given sample set. When using *SciPy*'s integrated *normaltest*, based on D'Agostino's K^2 test, we outcome a p-value of 1.192283989686776x10^{-71} which also indicates a rejection of our null hypothesis.

Based on our testing, the score distribution of our best performing ANT on Ms Pacman cannot be proven to be normally distributed. Because of this, we make no efforts to examine whether or not normal distribution is present for the SDT model.

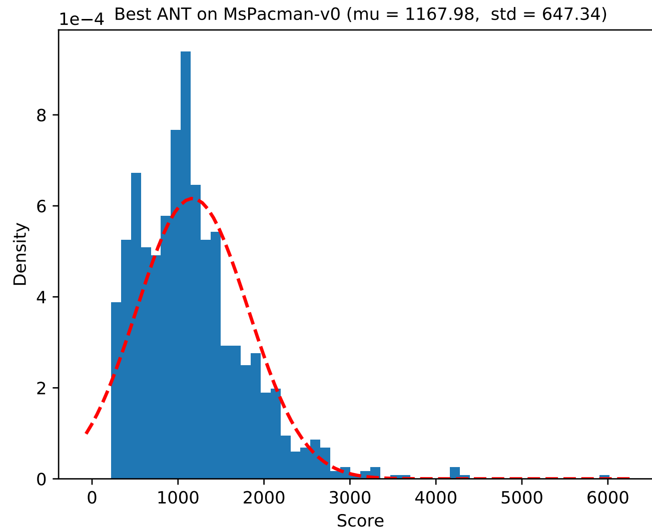


Figure 5.22: Distribution of achieved scores of the best performing ANT during 1000 completed game sessions. The best score is 6023 while the lowest is 230 points. The red dotted line is a Gaussian normal curve with the same mean and standard deviation as the distribution.

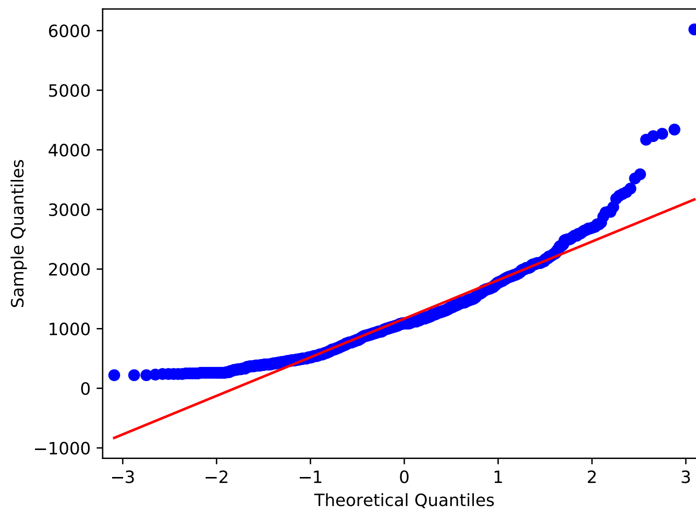


Figure 5.23: Quantile-Quantile plot of the distribution. The red diagonal line indicates normality in the distribution.

Now that we posses a best performing ANT, from now on called ANT_{best}, we examine the performances of trained SDT models to find SDT_{best}. We perform the same strategy as before with a scatterplot giving an overview of our trained policies (figure 5.24). We trained for depth

1, 2 and 3 each 5 models per dataset. Additional training is performed with D_1 where models of depth 4 and 5 are trained (figure 5.25). If we look at figure 5.21 we can observe a best performing $SDT_{limited}$, with a depth of 2 and trained on D_2. This SDT variant has a limited amount of nodes to equalize the computational power to the best performing ANT. The average performance of this model is 985,28 points. If we examine trends, we can observe a positive one for D_1 where the regression coefficient R_{D_1} is equal to 14,59664. A downward trend is observable for R_{D_2} with a coefficient of $\approx -11,34$. The trend for models from both datasets is slightly positive at ≈ 2.

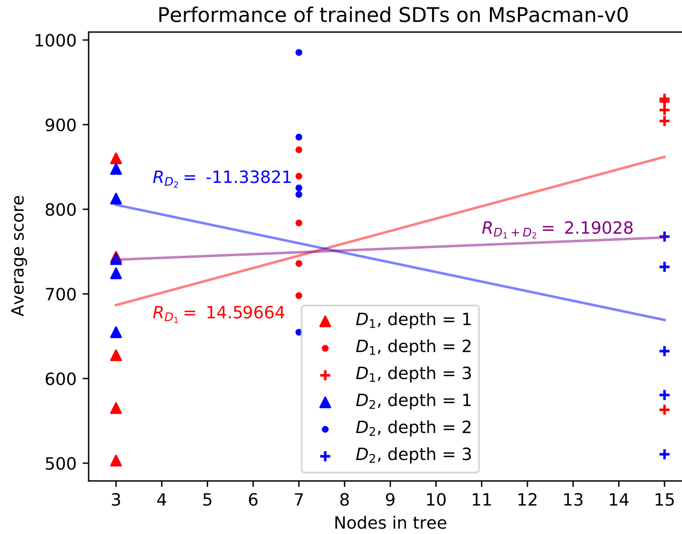


Figure 5.24: SDT models and their performance compared to their node complexity. Performance is the average of 5 sessions of 100 games in a row with random seeds. For D_1 the relation between performance and node complexity is negative, while D_2 is positive. The mean trend of both datasets combined tends to be slightly positive.

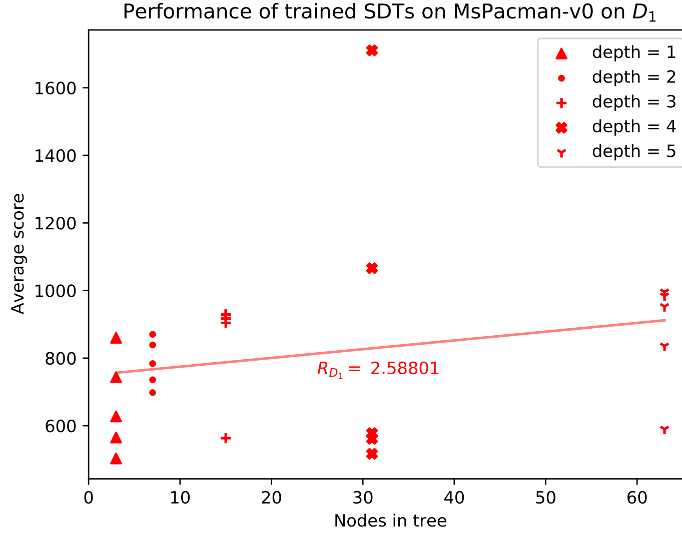


Figure 5.25: SDT models and their performance compared to their node complexity. Performance is the average of 5 sessions of 100 games in a row with random seeds. The relation between performance and number of nodes tends to be positive. The best performing SDT achieves an average score of ≈ 1650.

When looking at figure 5.25 we observe a positive trend of 2,58801 from our samples towards a depth of 5. One outlier at a depth of 4, with an average score of 1710,74 can be considered as SDT_{best} when deriving from our sample set of trained models.

The following histograms (figure 5.26 and 5.27) represent the distributions of rewards gained of different policies in a session of 10.000 games. In the first plot (5.26) we observe the score distribution of SDT_{best} to be between ANT_{best} and the original DQN policy. The mean of ANT_{best} is 1114,284 with a standard deviation of 594,4377, SDT_{best} 1664,523 and 674,6945 respectively. For the original DQN policy a mean is observed of 2345,317 with standard deviation 676,693. The largest frequency peak is for the ANT_{best} model with a frequency of ≈ 1100 for a score between 1000 and 1100. Here we conclude that the best performing policy, the one with the highest score on average, between our two tree models is SDT_{best} with a difference of ≈ 550 points. The spread of scores is lower for the ANT model, mainly because of higher slope between 0 and mean compared to the SDT which follows a more bell-shaped curve. We do remark that no conclusion could be made about the distributions whether they follow a Gaussian curve or not. The original DQN policy, where the distillation dataset is taken from still, outperforms with a difference of ≈ 680 points. When comparing maximum achieved scores, ANT_{best} got a score of 6180 points whereas DQN gained a score of 6260. This makes the gap between both models at their maximum potential smaller to a difference of only 80 points. SDT_{best} managed to get a maximum score of 5750 points, a difference of 510 points with the DQN and 430 with the SDT.

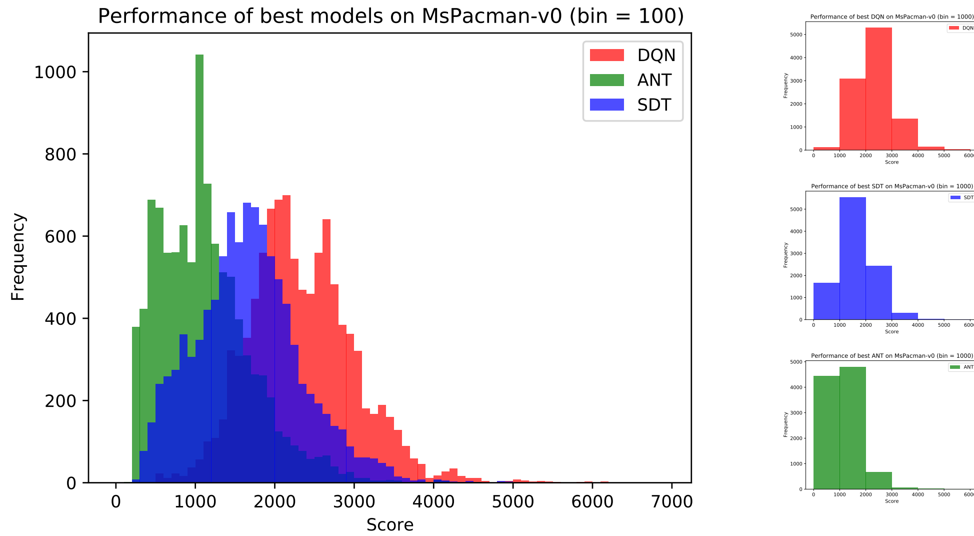


Figure 5.26: Score performance histograms of the best models from 10.000 played games. Left are The policies compared to each other with bins of size 100. Right are histograms of policies with bin size 1000.

Looking at the first histogram, we can conclude that SDT_{best} clearly outperforms ANT_{best}. This however is a comparison between a model of 31 nodes to one with only 5, giving more computational power to the larger SDT model. We want to examine on a more *node-efficiency* level the performance difference between the models. We do this by taking the number of nodes in our ANT_{best} and setting this as an upper bound for the SDT model to be considered for comparison. Going back to figure 5.24 only a SDT with depth 1 or 2 classifies as being comparable. Because most computation of the model happens in the internal nodes and not the leafs (these only contain logits compared to more complex perceptrons in the internals) we also include SDT models of depth 3 and thus 15 nodes in the structure.

From all the trained models up until depth 3, we chose the one with a best average performance of 985,28 points and labeled it as $SDT_{limited}$ (figure 5.24). This SDT is one with 7 nodes and thus depth 2.

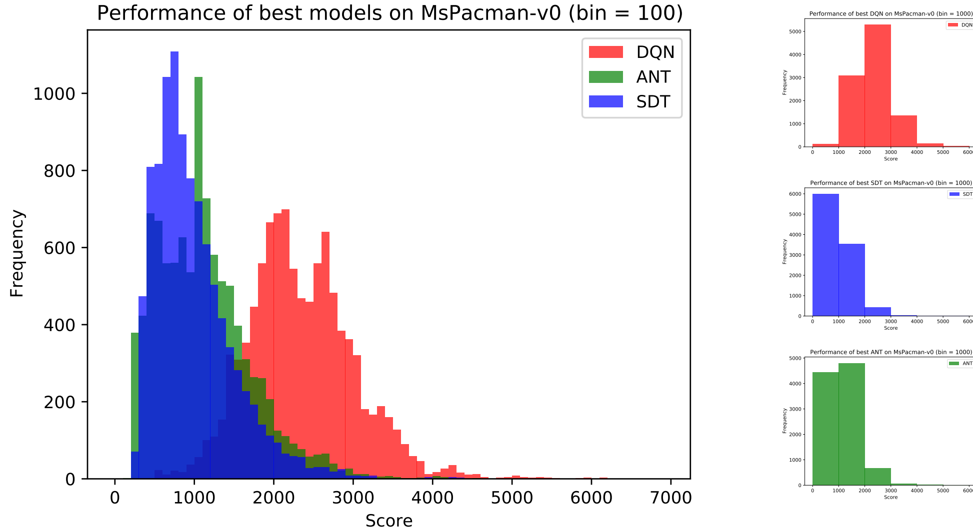


Figure 5.27: Same histograms but with limitations for the SDT model (visualising $SDT_{limited}$).

From the histogram of figure 5.27 we have the same DQN and ANT_{best} distributions as before. The new $SDT_{limited}$ distribution has a mean of 986,764 and standard deviation 510,8166. The maximum achieved score is 4660. We observe a difference of 127,52 between the two trees with the ANT achieving higher average performance than the SDT.

Doing these comparative experiments gave us enough insights to conclude that SDTs outperform ANTs when following our pipeline on Ms Pacman. The performance of the best SDT SDT_{best} is comparable to the results of the original DQN model. However, because SDT_{best} contains roughly 6 times the amount of nodes ANT_{best} has, it is biased in the amount of computation we give it. As a better comparison of efficiency, $SDT_{limited}$ is considered with the same depth as ANT_{best}. By doing this, we now observe ANT_{best} to achieve higher results compared to the SDT.

Interpretability analysis

As a second round of comparisons in our analysis, we now zoom in on the interpretability aspect of the trained trees. We consider all previous trees we named as well as a second ANT_{router} model who visualizes the internal routers instead of visualising the learned classifiers during training. This is accomplished by using our SRDS technique discussed in section 4.3. All visualisations are given in figures 5.28, 5.31 and 5.34. An enlarged version of the SDT_{best} visualisation can be found in the appendix. We begin the analysis by comparing the complexities (table 5.11). Note that we now only count internal nodes to calculate the complexity of our SDTs and not the leaf nodes. This more closely resembles the actual *visual* complexity of the figures shown. From the table we can observe an overall lowest complexity for $SDT_{limited}$ with a Φ_2^{avg} of 3. However, we could prune ANT_{best} to result in the same structure architecture since its second layer nodes only have 1 child. This is because ANTs learn their optimal structure without transformers so extra classifying nodes could be used to perform a routing similar to one with transformations enabled down the line.

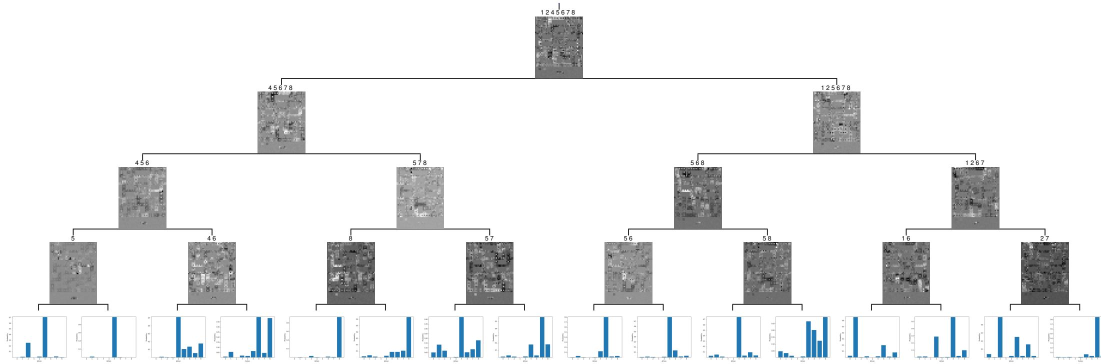


Figure 5.28: Tree visualisation of SDT_{best}.

Table 5.11: Complexity of the different visualised models.

Name	Min. path	Max. path	Avg. path	$\mathbf{\Phi_2^{min}}$	$\mathbf{\Phi_2^{max}}$	$\mathbf{\Phi_2^{avg}}$
SDT_{best}	4	4	4	6	6	6
$SDT_{limited}$	**2**	**2**	**2**	**3**	**3**	**3**
ANT_{best}	3	3	3	5,3219	5,3219	5,3219
ANT_{router}	**2**	3	2,6667	4,3219	5,3219	4,9886

With the complexity measured we can now focus on the insights individual nodes provide. We start with SDT_{best} and $SDT_{limited}$ and consider some examples. To remind the reader: the brighter the spots in the node, the higher the weights in that area and the more chance to be routed to the right side when pellets, ghosts or Ms Pacman is present there. The same is true for darker spots but for the absence of potential activation and a routing to left.

For SDT_{best}'s policy we observe the following behaviours:

- The agent runs away from ghosts when there are no dots nearby. When there are pellets behind the ghost, the agent will attempt to eat it and lose a life. In some situations, the agent got stuck for a few frames, allowing a ghost to attack it. This behaviour however is also present in the DQN policy and could be the result of the used prepossessing for the incoming environment frames.
- In contrast to the DQN model, the SDT_{best} agent does not go into the direction of the cherry when it is present on the screen. In the case of DQN, the agent would most likely go to the cherry when in the neighbourhood of around 10 pixels.
- Both DQN and SDT agents know the benefits of using the power dot, turning the ghosts vulnerable. DQN often takes the one in the bottom-right corner to start with while the SDT agent waits longer to take the bottom-left one later in the game. When the ghosts are weak, the agent sometimes know the benefits of eating a ghost and tries to chase them.

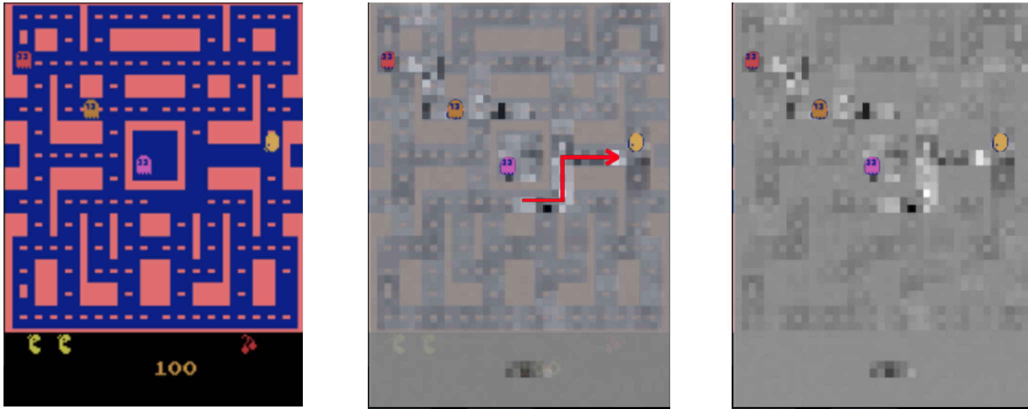


Figure 5.29: Game session played by SDT_{best}, visualising the leftmost node of the tree. Left: the environment after taking 12 steps with action 5 (up-right). Middle: same situation but with a partial overlay of the node weights and a path indicating the result of taking action 5 for 12 steps. Right: full opacity overlay of the weights over the environment.

Interesting behaviour can be seen with the SDT_{best} agent when starting a new game. For around 20 frames, the agent chooses action 5 (up-right) to go to the middle-right side of the maze while avoiding the ghosts who start by going to the top-left side. This behaviour is observable in the tree as the leftmost node resulting in both action 5 for each resulting leaf. The node is visualised with the sprites in figure (5.29). An indication of this behaviour being start behaviour are the black blocks at the score board. This could indicate that the agent looks at the score and performs this action on the absence of digits or in other words: when the score is still 0. If the agent loses a life, it will not follow the original starting policy, rather it will most often go in the opposite direction or downward. To substantiate our findings, we looked at the path chances at the beginning of a gaming session and for the largest part this node has the highest influence over the prediction of the whole tree.

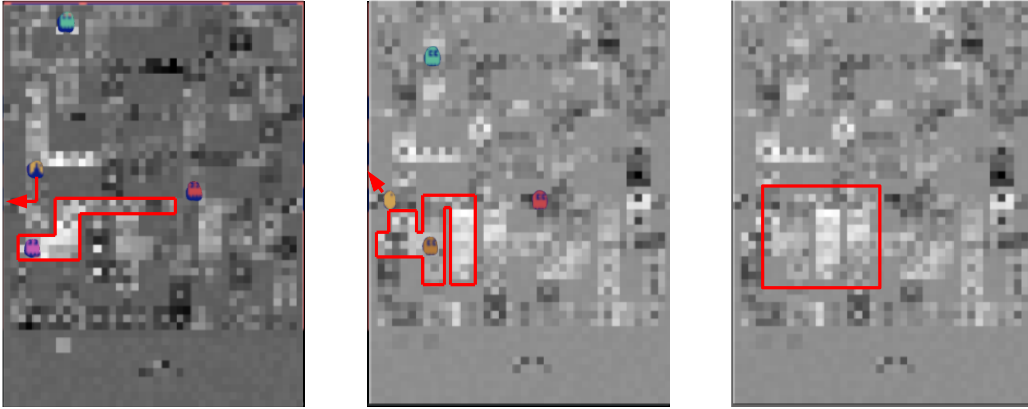


Figure 5.30: Visualisations of SDT_{best} nodes deciding on action 8 (down-left) and 6 (up-left). Left: An enemy is nearby the agent. It can escape by entering the corridor to the bottom-left and performs action 8. Middle: once in the corridor, the agent chooses action 6 to leave the space as soon as possible on the other side of the maze. Right: a composition of both weights for deciding action 8 and 6. We can observe a lighter area there where the enemies are closest to the agent, triggering it to flee the area.

Another node with strong indications is one deciding on action 8. This examples indicate the ability of the agent to escape danger by using the corridors that transport it from one side to the other (figure 5.30). Notice the absence of the pink ghost in the middle figure. This is due to the flickering behaviour of the ghosts on the Atari system.

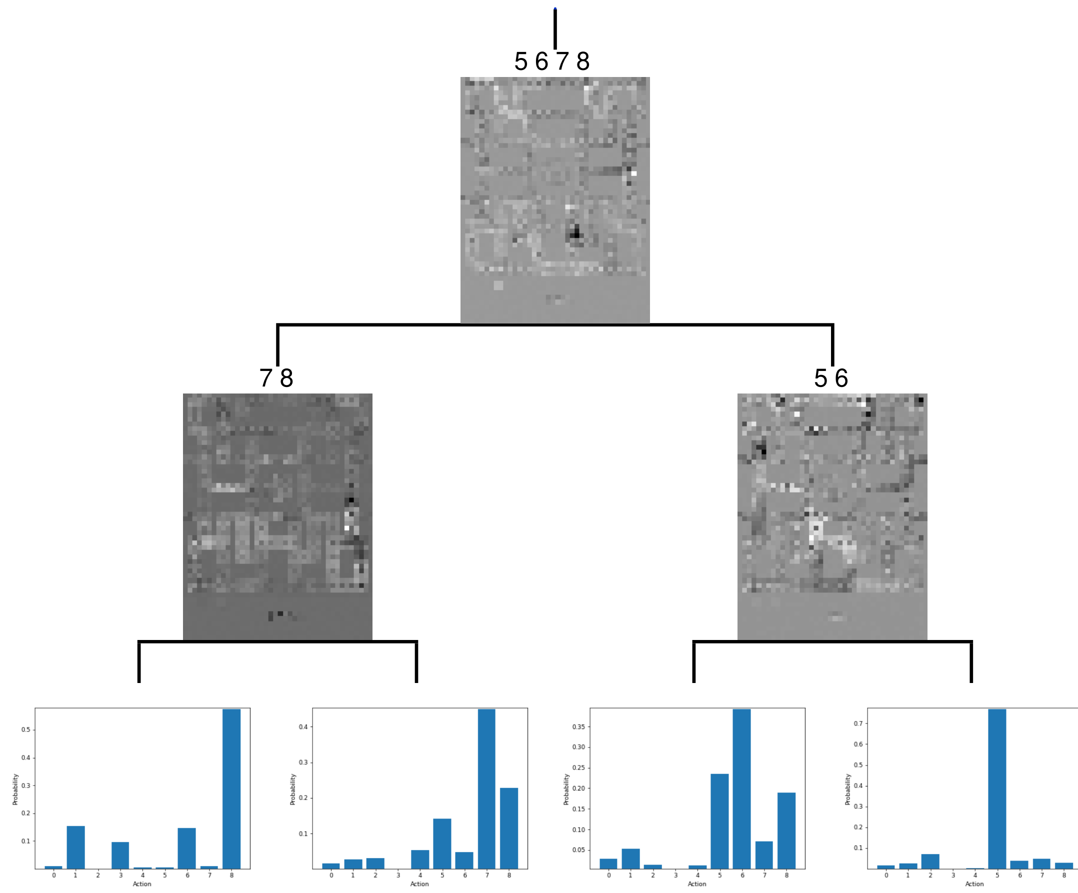


Figure 5.31: Tree visualisation of $SDT_{limited}$.

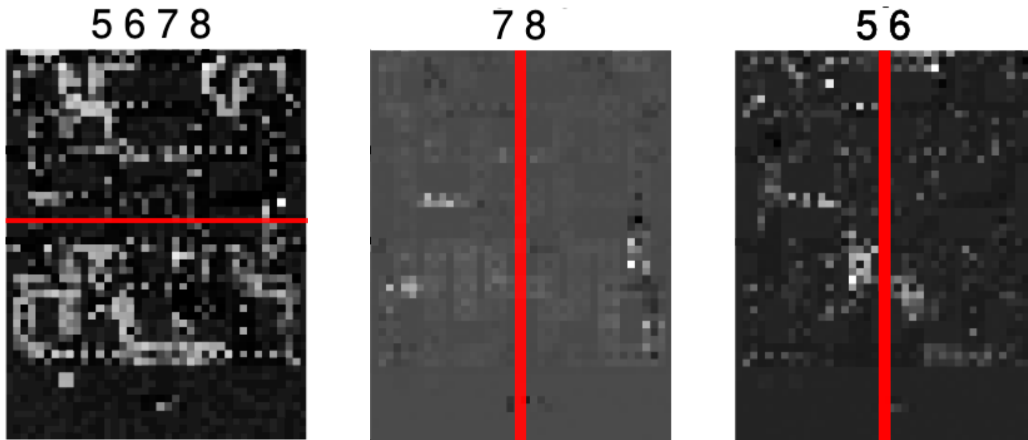


Figure 5.32: Nodes from $SDT_{limited}$ with higher contrasts to better distinguish sides. Left: the root node decides on actions going up or down. We can observe more bright areas on the under half of the screen, indicating more activity of enemies and the player. Middle: This node decides to go down left or down right. A clear divide between whiter areas left and right can be seen. Right: same distinction but with areas closer to the middle line.

When observing $SDT_{limited}$, we notice similar behaviour to SDT_{best} but with a lower gain in score on average. The visualisation of the model (figure 5.31) reveals a relation between the highest logits in the leaf nodes and the possible routing. Each supported action is a intercardinal direction covering all upper- and lower-bound actions. If we look at the leaves of the tree we can observe mostly deterministic distributions over the logits. If a path with high reaching probability is made towards a leaf node, it means the action of that node would greatly contribute with its highest logit to the total prediction of the tree.

The left side of the tree, at second level, routes most likely to action 7 and 8 (down-right and down-left) while the right side routes to 5 and 6 (up-right and up-left). This means that in the first node a decision is made to ether go up or down in the maze. The second layer decides to go left or right. Indication of this in the visualisation of the nodes (figure 5.32). The tree does also emphasise elements on screen like the number of lives left and the score, as can be seen in figure 5.33. If all lives are still available, then the agent would have a higher tendency to go down. To differentiate between going down-left and down-right, the absence of certain digits in the score is considered, indicated by the black blocks. These behaviours are in line with made observations.

Figure 5.33: Score and life indicator importance for $SDT_{limited}$. Left: the root node takes into account how many lives are available to the agent. Right: after the root node decides to go down, the score is checked on the absence of certain digits.

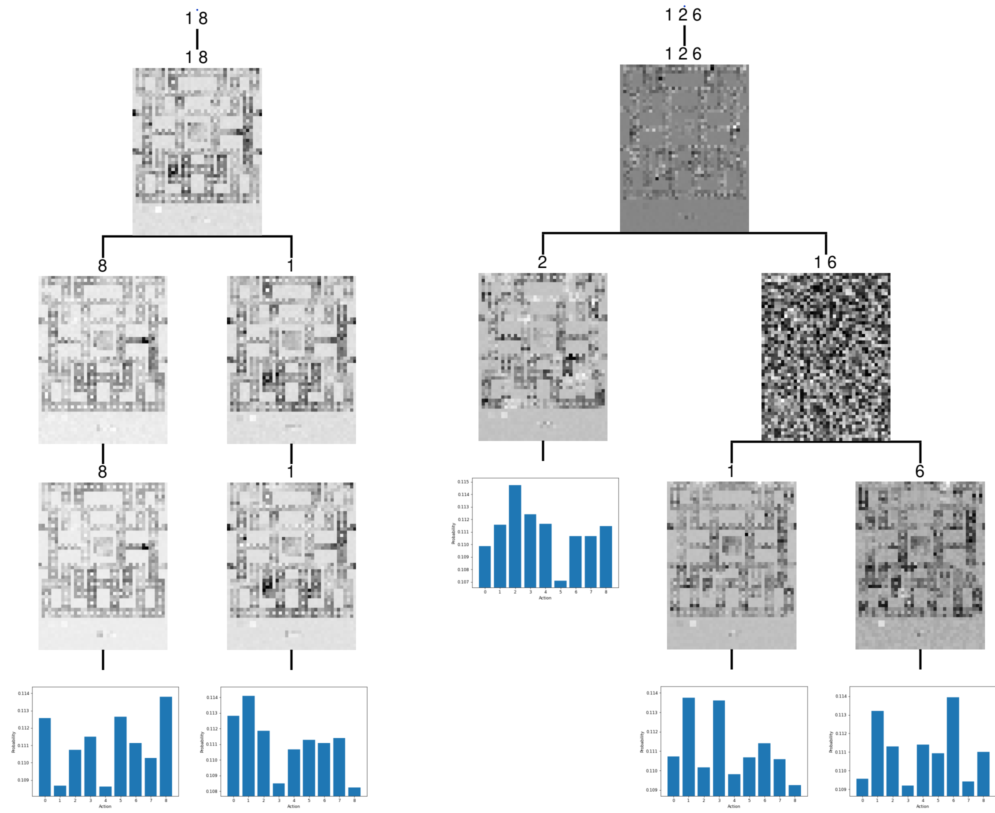


Figure 5.34: Visualisation of ANT_{best} on the left side and ANT_{router} on the right side. ANT_{best} is visualised with the learned classifiers during training. ANT_{router} visualizes the actual routers when using the SRDS technique.

Now that we have examined the SDTs, we now focus on the ANTs. The following observations in behaviour for ANT_{best} are made:

- Behaviour when starting a game is almost identical to SDT and the original DQN. Difference here is the tendency to stay on the bottom side of the maze, collecting the first power dot in the down-right corner.
- The agent has the tendency to run away from ghosts, but often turns around and gets killed by the pursuing enemy.
- In corners the agent can get stuck doing nothing. This is more noticeable than the other policies. Getting stuck also happens when it returns from going into one direction when hitting a corner.
- More often than normal the agent hesitates to take a power dot. Even in situations when the agent could clearly benefit from consuming the dot while being chased, it will likely refuse the opportunity to take it.

If we look more closely at the visualisation of ANT_{best} in figure 5.34 we see less distinctive areas of dark and bright compared to the other models. This could be the result of visualising the classifiers instead of the routers in the internal nodes. Classifiers have a fully-connected linear layer with input size the incoming environment frames and as output the amount of possible actions. This is different from normal routers who have layers from input size to only 1 output, deciding on the routing left or right. In the case of Ms Pacman with 9 actions there is a 9-fold in the amounts of weights in the layers. That is why we sum all of the 9 weights per action to give the total weight activation of all actions, which results in this more ambiguous node visualisations. It is therefore difficult to reason about what the internal nodes decide upon. Furthermore, contrast is reduced because of higher average weights over the entire environment. We observe more uniform action distributions at the leafs compared to the ones from the SDTs.

One solution to solve the problems with routers is to learn the tree with the *smart routers, dumb solvers* (SRDS) approach as mentioned in section 4.3. This results in the ANT_{router} model (figure 5.34) where the internal nodes are visualised by their routers instead of the classifiers. However, this comes at a cost of lowering average performance. Another approach is to visualise ANT_{best} per weight vector of a certain action. This makes recognizing important regions for an action easier to distinguish per class. We opted to both visualise the SRDS model ANT_{router} and a version where each action in the classifier has a different visualisation, leading to 9 SRDS ANTs (one per action). Two of them, representing action 7 and 8, are given in figure 5.35.

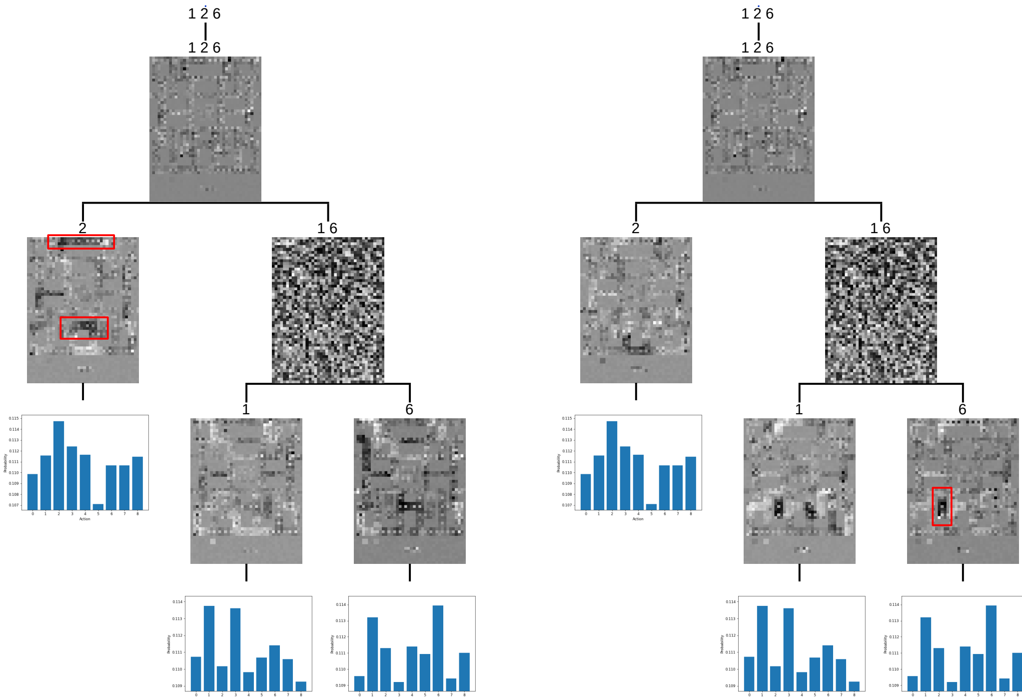


Figure 5.35: ANT_{router} with only visualising one action in the solver leafs. Left: action 7, going down-right (ANT_{router_7}). Right: action 8, going down-left (ANT_{router_8}). Important distinct areas are indicated with red.

All ANT router variants have in the second layer on the right a router node with random noise in. This is because this router is not trained properly during the learning process. Its routing behaviour closely resembles an uniform distribution over the two possible ways to go. It could be pruned but then the root node would have three children, meaning it is not a binary tree anymore. For ANT_{router_7}, responsible for the down-right action, we see black areas at the center and top of the maze in the middle of the screen when we go left-down in the tree. These regions indicate absence of any enemy or player activity, meaning the agent is more decisive to do action 7. If we look at the logits in the plot, we see a low values for action 7, meaning a lower chance for 7 being the prediction of the tree even if the node mostly matches with the environment state. The same reasoning can be done with ANT_{router_8} where on the bottom-right side of the tree both leaf nodes show a distinctive black area on the middle left side of the node. Again, if the state of the environment mostly matches with the weights of action 8, its support to the decision as a whole would be low since the logits have low values for that action.

We have proven that the experimental pipeline is applicable for the Ms Pacman Environment. Which one of the two types of models performs best depends on the amount of nodes allowed in the structure. When no limits are stated, the SDT models outperform ANT models. If the number of nodes is limited (or equalized) then ANTs gain better performance. For the interpretability part we can distinct several insights in the models that could explain its behaviour in the environment. We now repeat the same experiments with the Enduro environment.

5.3.3 Enduro

In this racing game the player has to catch up to other racers from last place up to pole position. The game is divided into several days. Each day a certain number of cars has to be surpassed in order to progress. The objective is to pass 200 cars on the first day, 300 on the third and so on. Visibility and steering behaviour changes over time to make driving the road more difficult. Fog can make the visibility of the racers more difficult while randomly placed ice can make the road slippery. In OpenAI Gym, the possible actions are going forward (action 1), steering right and left (action 2 and 3), going left and right (7, 8) and doing nothing (0). Actions (4, 5, 6) are for applying breaks straight, while steering right and while steering left.

Momentum is maintained when at a certain speed and the gas button is released. To slow down, the player has to apply the breaks. This is necessary in slippery situations such as on ice.

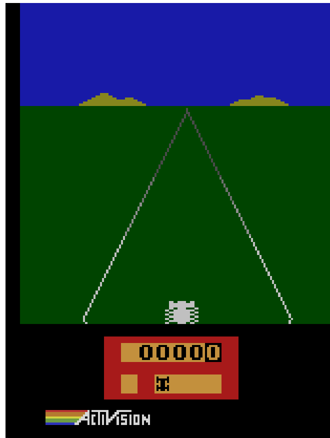


Figure 5.36: The Enduro environment.

Policy selection

We trained 33 different policies on the Enduro environment: 15 DQNs, 9 A2Cs and 9 PPOs. From the boxplots given in figure 5.37 it is clear that the most performant policies are generated using the PPO algorithm. The other two types achieved only a maximum score of 8 per type. This could be explained by the difficulty of the terrain and its varying environment throughout the play. PPO is more suited for environments involving high risks like the slippery ice areas and the misleading fog. The best performing PPO policy on 100 games is a outlier with an average score of 210,96.

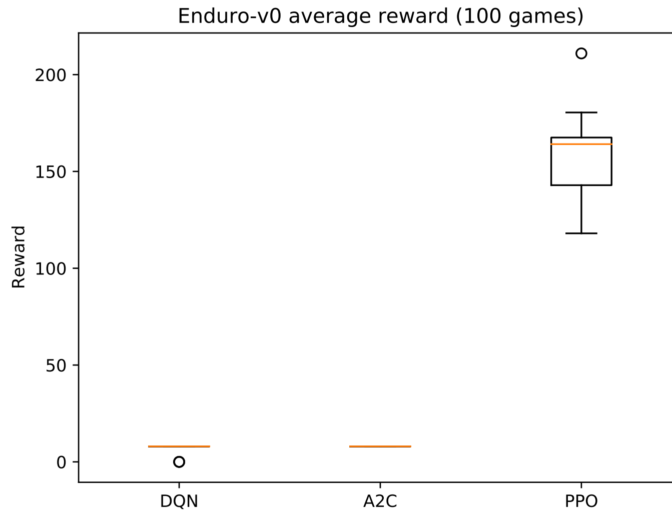


Figure 5.37: Boxplots of the trained Enduro policies and their performance on 100 games.

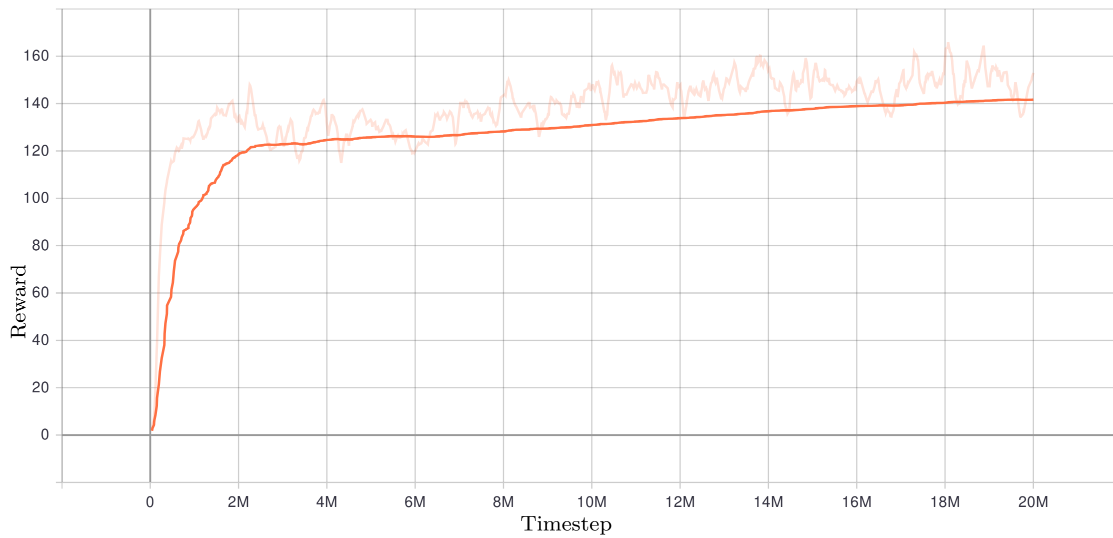


Figure 5.38: Mean reward curve of the best PPO agent.

Knowledge distillation

For Enduro we only generated one size of dataset containing 100.000 frames. This is a set from a selection of 10 different recordings which gained the highest commutative score during generation. D_1 contains 100.000 frames comprised of 88 games and a average score of 202,47.

Performance analysis

From the experiments done, the best found performing ANT consists of a single solver node (figure 5.39). It gains a an average score of 35 points and is trained with 50 episodes of growth and 50 finetune (F50/G50). The dot representing this model is the highest in the scatterplot. The coefficient of our regression line is -0,68198 when applied to the scatterplot of all 30 trained trees.

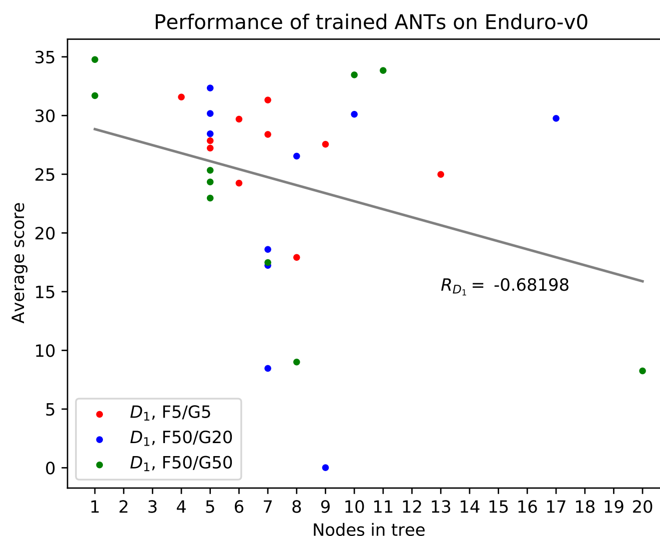


Figure 5.39: ANT models and their performance compared to their node complexity. Performance is the average of 5 sessions of 100 games in a row with random seeds.

The best performing SDT, SDT_{best}, out of the 45 trained is one of depth 5 (figure 5.40). We could have trained more complex trees, but as our coefficient is only 0,0562 we do not expect much gain in score when increasing the depth of the model. Some outliers are observable for depth 2 and 3 with a value around 11 points. SDT_{best} achieves an average score of 28,5363.

If we examine the histograms of figure 5.41, we can observe an indication of higher performance for ANT_{best} compared to SDT_{best}. The average gained score of the SDT is 28,784 with a maximum of 94. This is lower than the ANT, which has a mean of 33,377 and a maximum score of 108 points. In comparison to the original PPO model, both trees perform noticeably weaker. The DRL policy gained a score of 202,893 on average and achieved a maximum of 416. This performance could be explained by the changing colors of the environment due to the day and night cycle. The learned tree models could be confused by the preprocessed images that also disregard the changing of the environment when applying thresholding. A better approach would be an adaptive threshold based on the situation happening in the environment.

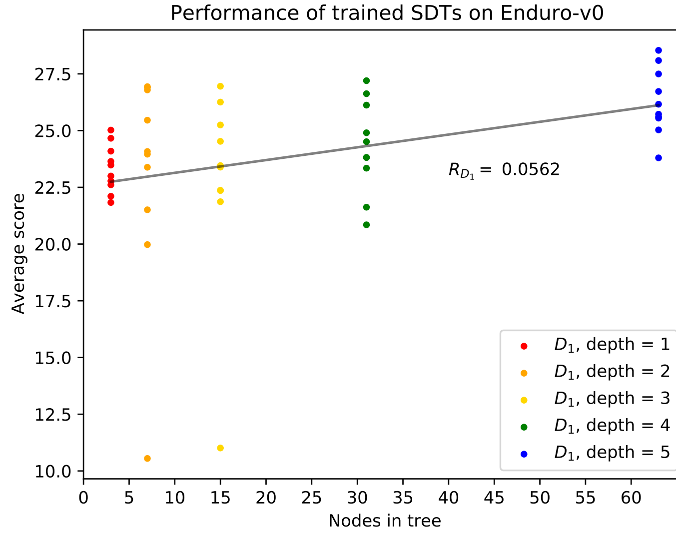


Figure 5.40: SDT models and their performance compared to their node complexity. Performance is the average of 5 sessions of 100 games in a row with random seeds.

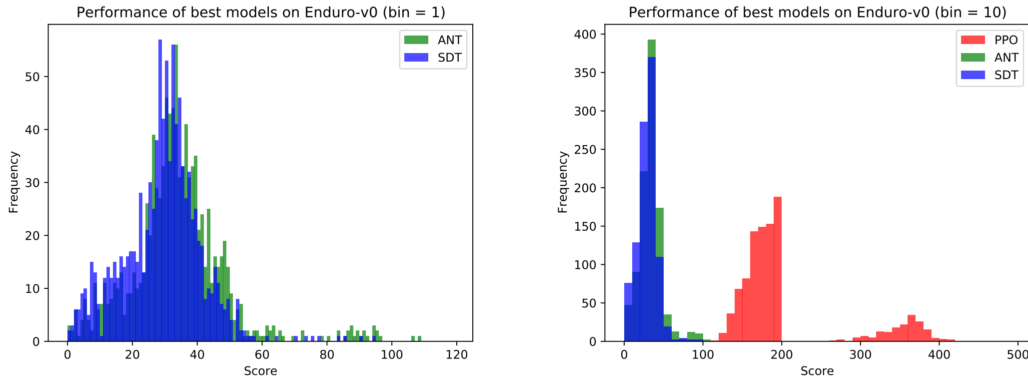


Figure 5.41: Score performance histograms of the best models from 1000 played games. Left are the tree policies compared to each other with bins of size 1. On the right plot the original PPO policy is included with bin size 10.

Interpretability analysis

When looking at the composition of the trees, we see that ANT_{best} has only a single node while SDT_{best} has 64. The visualisation of SDT_{best} is given in figure 5.42 with a larger view given in the appendix. ANT_{best} is visualised in figure 5.43. Because the simplicity of the solely classifier in the ANT, we also visualised the weight for each of the 9 possible actions (figure 5.44). Because no routers are included in the structure, the need for the SRDS technique is obsolete.

From the logits plot of ANT_{best} we can see lower values for action 5 and 6, meaning throttling down while steering left or right is a not so common strategy. These actions are clearly distinctive

in figure 5.44 where the weights are trained on less frames resulting in clearer weight maps. It is also hard to find meaningful interpretations for the single node visualisation of the solver. The only possible insight we could make is the importance of the regions at the border below the road. The middle has a darker area while the left and right side are brighter. This could indicate a reliance on these areas when the agent tries to pass by a competing player's car. Overall the ANT has visualisations that are not entirely interpretable. The SDT's visualisations are also not easy to interpret but offer more distinct features in the resulting maps.

We finally can conclude that in the Enduro environment ANTs gain higher performance compared to SDTs with lower amounts of nodes in the structure. In fact, they are as simple as possible, containing only one node. However, when looking at the provided insights, SDTs still provide more information. We encourage the reader to find further possible interpretations of these models.

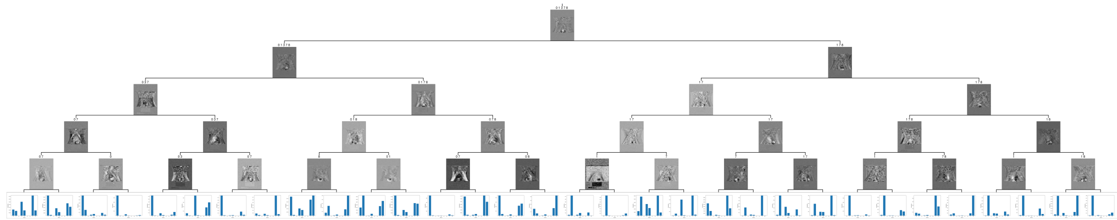


Figure 5.42: Tree visualisation of SDT_{best}.

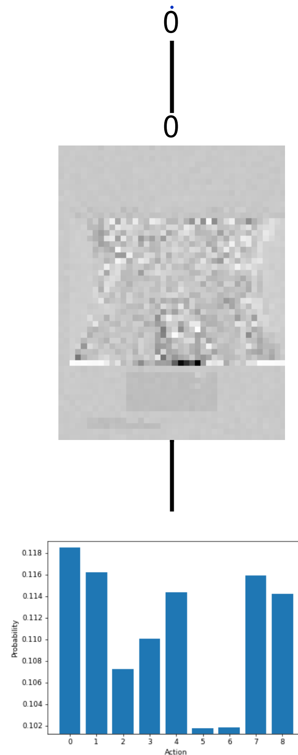


Figure 5.43: Visualisation of the only solver node in ANT_{best}.

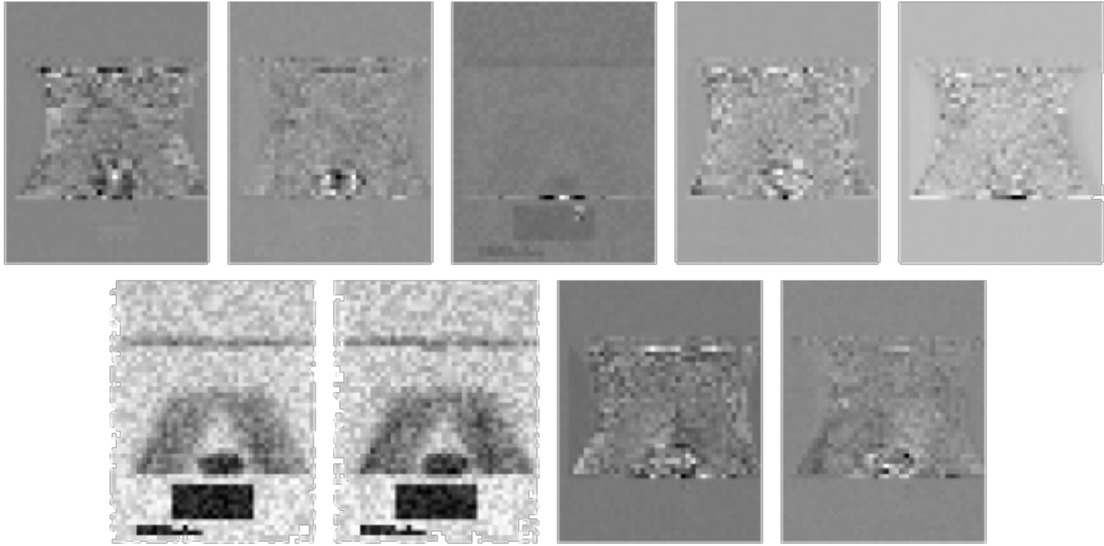


Figure 5.44: Each frame is from an action of the ANT_{best} solver (above the first five actions, below the last four.)

Table 5.12: Complexity of the different visualised models.

Name	Min. path	Max. path	Avg. path	$\mathbf{\Phi_2^{min}}$	$\mathbf{\Phi_2^{max}}$	$\mathbf{\Phi_2^{avg}}$
SDT_{best}	5	5	5	10,9773	10,9773	10,9773
ANT_{best}	**1**	**1**	**1**	**1**	**1**	**1**

5.4 Results

We briefly summarize the findings from our experiments and what this has to contribute to the thesis topic. We will mainly focus on experiments done within the Ms Pacman environment since it could provided most of the analysis we could do in this evaluation.

In our first experiment we trained vanilla SDT and ANT without any changes to the code. This was to subject the models to a commonly used benchmark and to compare their performances on both interpretability and performance. We trained several models per possible parameter: per depth for the SDT and per amount of grow/finetune for the ANT. We found that, using the parameters given in the respective paper, SDTs deliver higher accuracy performances compared to ANTs. This however comes at the cost of being exponentially more complex in the number of nodes the more layers we add to the SDT architecture. The differences between the most accurate ANT and SDT are significantly low, with the best SDT only performing 1,33% better in accuracy than the best performing ANT. For this difference, the SDT needs almost 7 times the amount of nodes making it more complex in a interpretability context as well as needing more computational resources. We could measure this visual complexity with our created indicator, confirming our claims that a best performing SDT is more complex.

For our in-depth analysis in a XRL context we followed our proposed experimental pipeline for a first time on the Ms Pacman environment. There we distilled from a best performing DQN model several datasets to train different tree models with. At the analysis part of the pipeline we compared the found models on both interpretability as well as performance in-game. First we tested several parameters settings of the ANT to see which ones are the most optimal. These were ANTs with deterministic routers and where soft decisions were made. We then first analysed our trained ANTs to find the highest performing one. Afterwards we considered the possibility of the gained scores of the best models playing to be normal distributed or not. This was not the case according to both visual and statistical tests. We analysed with the same strategy our trained SDTs and found a best performing one as well as a best model with a limited amount of nodes.

Comparing the best models was done using an average performance test over 10.000 games where distributions of the average scores could be made from. From this we concluded that a best performing SDT achieves higher scores than an ANT. Once again, we acknowledge the fact that the SDT has a significantly higher amount of nodes compared to the ANT demanding more resources and making it more visually complex. We compared the models with a limited SDT model with a similar amount of nodes to the ANT. From that we could conclude that an ANT outperforms an SDT for roughly the same amount of nodes.

At last we did an extensive visual analysis of all found best models. Here we brought forward some insights that the models could provide. We do notice that it is more difficult to hypothesise the meaning of nodes in an ANT because of the complexity in their solvers. This we tackled on with our SRDS approach but still managed to get less descriptive conclusions from the visualised weight maps.

For a second iteration of the experiment, on the Enduro environment, we did the same performance testing and visualisation. There we found out that a simple node ANT structure outperformed the best playing SDT with 63 nodes. Interpretability-wise the ANT was of course less complex according to our metric but the made observations indicate that the SDT has more distinct features that could be distinguished from the weights map.

Chapter 6

Conclusions

In the final section of this thesis we state our conclusions made surrounding the topic and experiments done, overview the made contributions, discuss the results and how they were achieved and look towards possible future work.

From the made literature research we can state that the field of explainable artificial intelligence (XAI) is in an early state yet we acknowledge the broad history forgoing the field on making machine learning models more interpretable. For six important economic sectors (logistics, healthcare, legal, defense, financial and industrial) we discussed several applications and potential use cases for future XAI integration. The examples given are situations where we could see benefits within the next 10 to 20 years while the field matures. We conclude that most taxonomies used in XAI literature are closely related to one of the three types of box models there could be: white, grey and black box models. As an even more recent field, explainable reinforcement learning (XRL) is a merger between XAI and the field of RL. Because the popularity of deep reinforcement learning (DRL), the need to explain black box models like deep neural networks only increased over the years. Techniques from XAI can therefore be used to 1) explain certain behaviours from an RL agent, 2) give insights into the black box models or 3) train or substitute with an surrogate model that is inherently interpretable.

We proposed a combination of two types of box models, white and grey box models, to be used with the upcoming knowledge distillation technique to make DRL policies more interpretable: weight map visualisations in combination with a tree-based model. For the models we opted to use a Soft Decision Tree (SDT), because of its proven ability to be a good surrogate model, and an Adaptive Neural Tree (ANT), which learns its optimal structure during the training process rather than being static from the beginning. We compared both models on several tasks. The first one was on the MNIST dataset benchmark where trained SDTs performed only slightly better than ANTs on prediction accuracy. However, if using the same amount of nodes as the ANT, the SDT underperformed. Because of the high number of nodes, the visual complexity of a SDT was much higher than an ANT. We conclude that an SDT is a better MNIST classifier but for a very small accuracy trade-off of around 1% we can make the model much more interpretable by using an ANT.

When comparing both models in a XRL context (playing Ms Pacman in the OpenAI gym environment), we can state that a SDT gains higher performance in scoring. We note however that like in the MNIST case the number of nodes in the best SDT is substantially higher than in

the best trained ANT, giving more computational power to the SDT. When comparing the best ANT to a limited version of the SDT, we can conclude that a ANT gains higher performance if the SDT has around the same number of nodes. On the interpretability side, if we compare the best models, the visual complexity of ANT is lower than SDT. However, if we try to equalize the number of nodes between the models, then the SDT gains a lower complexity measurement. One difficulty of ANT is the way the nodes are visualised and therefore interpreted. In the original model we visualised the classifiers, giving a more complex and actually incorrect insight into the routing behaviour of the internal nodes. We solved this issue with *smart routers, dumb solvers* (SDRS) but still had to rely on a hybrid version where the solvers's weights need to be visualised per possible action. This makes the needed number of trees to be visualised equal to the number of actions, which further complicates interpretation. Combined with their non-deterministic action distributions, we can conclude that ANTs are more complex than SDTs in their provided insights but can be on a per-action base more interpretable due to their reduced structure. When it comes to giving insights into the RL agent's behaviour by providing an inside look via the node visualisations, a SDT can provide more information than an ANT.

When the experiments were conducted in the Enduro environment instead, we concluded that the best trained ANTs outperformed the found SDTs. This however was at the cost of a having significant worse performance compared to the original PPO model the trees were distilled from. SDTs were still more complex than ANTs but gave better insights in their learned features compared to ANTs.

6.1 Contributions

We made four new contributions to the field of XAI and XRL with this thesis. In the literature research we did a comprehensive study into commonly used XAI techniques and some emerging and state-of-the-art methods. We provided a categorization of techniques based on the most recent works of XAI overview literature that mostly resembles the different kind of box models. *Inherently interpretable models* are synonymous with white boxes while *post-hoc representations* give more insight into the black box by graying it out with meaningful information. The third category is on itself not based on a black box, but rather tries to provide explanations to a black box using secondary mechanisms. The naming of the classes, their definitions and compositions are open for discussion.

Because of the lack of a metric for visual complexity in trees in scientific literature, we developed our own measurement. Partially based on computational complexity theory, we stated that perceived complexity both arises from a global and a local view of the tree. This is both the tree structure as a whole as well as an explanatory path from root to leaf respectively. For the global view part we chose an logarithm with base k to combat exponential growth in the number of nodes for a k-ary tree. The path is analog to traversing a list of nodes, so its complexity would be the number of nodes in that path. The summation of both *structural complexity* as well as *explanatory complexity* gives us a usable indicator for any kind of k-ary tree. We demonstrated that the variant using maximal path length is not injective, so we proposed to use the average path length to create a more unique measure.

The *Graybox* prototype framework could be considered as a key contribution to be able to conduct the experiments we had in mind. It has been extensively used throughout the thesis to train, analyse and develop the many models and datasets needed to test both tree types. The main three additions provided by the framework are its complete KD process based on our

experimental pipeline, its ability to create visualisations of different types of trees and the web interface. For the comparison of ANTs and SDTs, the KD process required us to facilitate the different versions of sets and experiments we wanted to set up. By introducing the profile-based storage and clear naming convention, creating many different experiments was uncomplicated. Its command-line interface (CLI) also facilitated the execution of experiments on resources managed remote machines using the terminal and a SSH connection. Code for the visualisation is provided within one module file and is adaptable for other types of tree structures implemented in PyTorch. With ETE3, it is easy to change the layout of the tree to the desires of the user. Finally the web interface provides an interactive environment between the user and the agent. The functionality of running OpenAI gym in a browser can be used to showcase the behaviour of the tree or DRL agent. Combined with the visualisation of the policy using the tree view with path indicator, a simpler manner is provided to study the policy's actions for both experimental and demonstrative showcase.

Finally, we provided extensive MNIST benchmarks for both different types of ANTs and SDTs. With the resulting data we compared the two and took conclusions on both interpretability and accuracy performance. The same comparisons were done with games of the OpenAI gym with an in depth analysis of the Ms Pacman environment.

6.2 Discussion

The literature review is done using a broad range of publications. However, we note that for creating our XAI categorization we only relied on four papers and one book. For a more extensive literature research the amount of publicised works should be higher. An improvement could be a broader coverage in publication time of the papers. We only discussed overview papers from the past 3 years. Whereas XAI is a young field, it is build upon a rich history of interpretable machine learning techniques that forms a significant basis.

There could be disagreement over the classification of certain techniques into our taxonomy study. Techniques like *model distillation* could both be classified as a *post-hoc representation* technique as well as a *additive explainable model* depending on the resulting model the policy is distilled into. This can both be a grey box model or a secondary model producing explanations about the original model. The relation between the categories and the types of box models we made could be questioned. The most significant property of white box models is that they are inherently interpretable by definition. Grey boxes give some insights into the models that were previously not easily observable, so they represent details in a post-hoc manner. The third category does not explicitly result in black box models, but rather uses a mechanism that would produce written explanations describing the behaviour of the models. We could state that the explanations themselves are white box models like rule lists or could be even more ambiguous with limited rule extractions. Therefore the analogy between a black box and an explanatory mechanism is not entirely correct.

For the MNIST dataset benchmarking we only used a sample of 5 trained trees per parameter configuration. This could be increased to a more significant sample size that would better describe the population. We also didn't compare models based on more statistical indicators like *recall*, *precision* and *F1 score*. We note however that these different performance measurements are partially irrelevant in the context of RL, which is the ultimate application of both models for this thesis. If the focus was more DL-related, then these indicators would have been more relevant.

The same small sample size has been used when training models for the XRL setting. However we did combine several models with different parameters in our scatterplots for the analysis of the datasets and possible trends. The negative trends when more nodes are present in the tree are at first sight counterintuitive in that we expected an increase in performance. This could be the result of the small amount of trained models. Our goal was to find a *best* model of each type and not to correctly analyse the population. We did a normal distribution test on only one metric, mainly the resulting performance of the best ANT on playing 10.000 games of Ms Pacman. It would be interesting to know if every performance distribution follows a normal distribution or not.

For all experiments done using a sample of models, they are still open for more experimentation and analysis. Permitted the resources, a broader more generalized study could be conducted where the testing result would be more significant.

6.3 Future work

The field of XRL, built upon the broader domain that is XAI, is relatively young. As the field grows over the years, we expect more and more XAI methods to be applicable into a RL context. Agnostic and intrinsic post-hoc interpretability-enhancing techniques are already well known to work within RL problems, yet methods that provide insights during training are not as numerous. Because of the importance for debugging and controlling the agent, more focus should be laid on developing and adapting ad-hoc (model-specific) methods.

Graybox, together with its interactive web GUI, could be used to train a broader range of policies together and provide tools to derive explanations from its visualisations. Its interface is basic but expandable to give more control to the user like rewinding sessions and frame storage features. With the framework, it is feasible to chose another (white box) surrogate model as the target structure to perform knowledge distillation on. The model-view part of the web interface should allow for an easy integration of such models if the accompanied visualisation library is provided.

As of today, most literature surrounding XRL involves toy environments like Atari games. For the field to mature further, techniques for training policies for usable applications like machine control and productivity enhancing should be developed in order to make them more useful as a real tool. An example would be the explanation of behaviour in robotic control done in Mujoco, a realistic physics simulator. For it, we lack the visual input like in our game examples since the environments only provide variables for instance joint angle and position to the agent. However this could be visualised with additional visual information such as the inclusion of the current state of the robot in a small figure. We don't know the limits of the knowledge distillation technique, but believe that it has huge potential to become a powerful method within XRL.

Appendix A

Graybox

Command Line Interface

Optional arguments are denoted with `(argument)` while expected ones are noted with `[argument]`.

Info

```
gray info
gray help
```

Display information and commands available in the application.

Profile management

```
gray profile-create [profile]
```

Create a new profile.

```
gray profile-set [profile]
```

Set currently using profile so repetition of the `profile` argument in other commands is not necessary.

```
gray profile-delete (profile)
```

Delete the current profile or another given one.

Train

```
gray train (profile) [environment] [algorithm] [steps]
```

Train a new policy with the training algorithm `algorithm` in environment `environment` and for `steps` steps. The result will be saved under the name `environment_algorithm_steps.zip`.

Generate

```
gray generate (profile) (environment) (algorithm) [model] [steps]
```

Generate a dataset of state-action pairs while playing `steps` steps in the environment. Because of the used naming convention of the policy file, information contained in the file name like environment and algorithm are not needed as parameters in the call.

Distill

```
gray distill (profile) [dataset] [tree_type] [args_file] (parameters)
```

Distillate a DRL policy into a tree-based model. At the moment of writing, the only values for `tree_type` are `SDT` and `ANT`. `args_file` is the path to a argument parser file containing the basic parameters for the model to be trained. Additional parameters can be given/changed with the `(parameters)` option.

Web

```
gray web
```

Launch the Flask server and provide the web GUI in the browser.

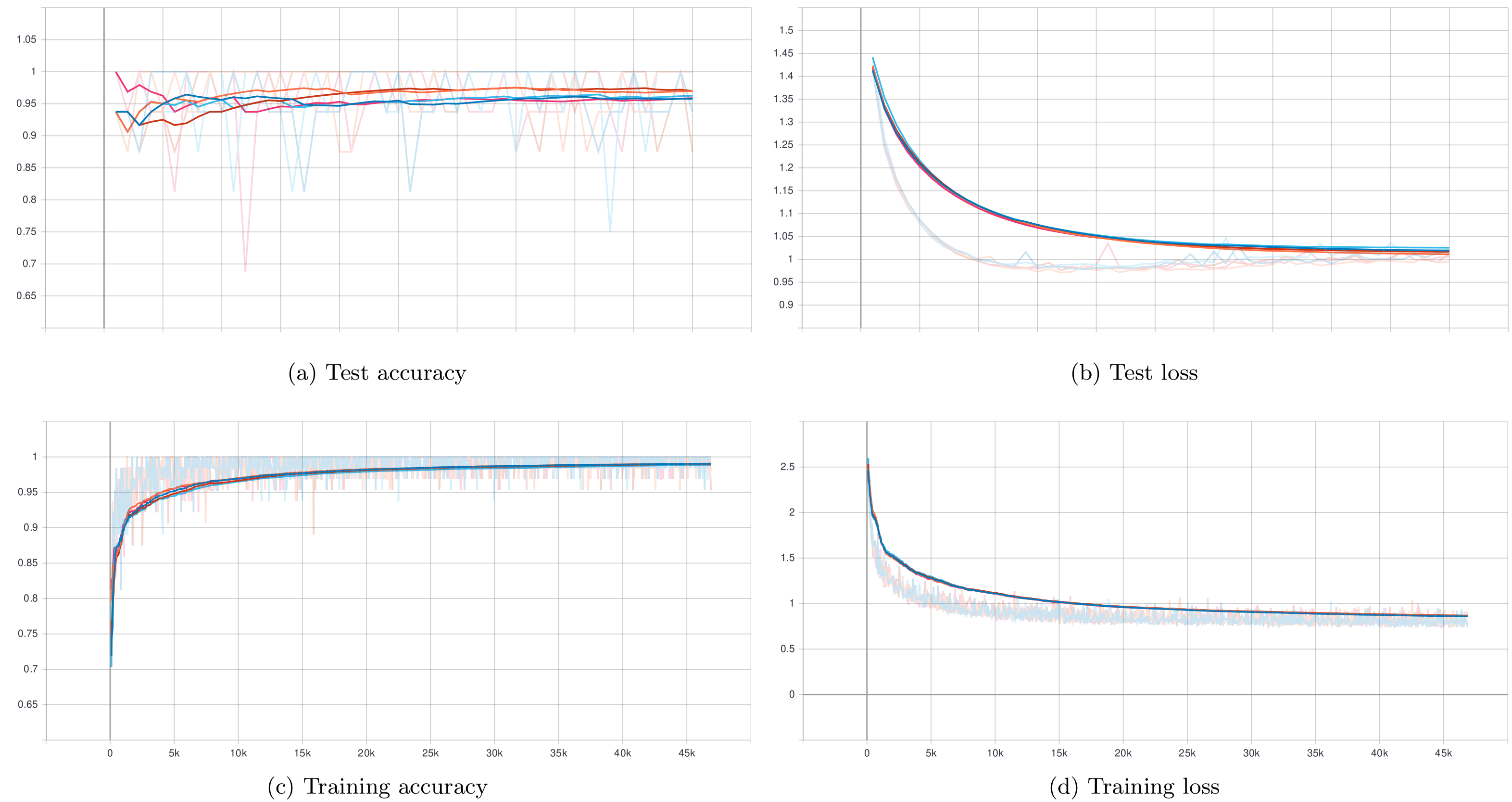
Code repository

Source code of the framework is available at `https://github.com/SenneDeproost/Gray_box`.

Appendix B

MNIST

SDT depth 9



(a) Test accuracy

(b) Test loss

(c) Training accuracy

(d) Training loss

Figure B.1: Experimental results for SDTs with depth 9

ANT 20 growth 100 fine tune

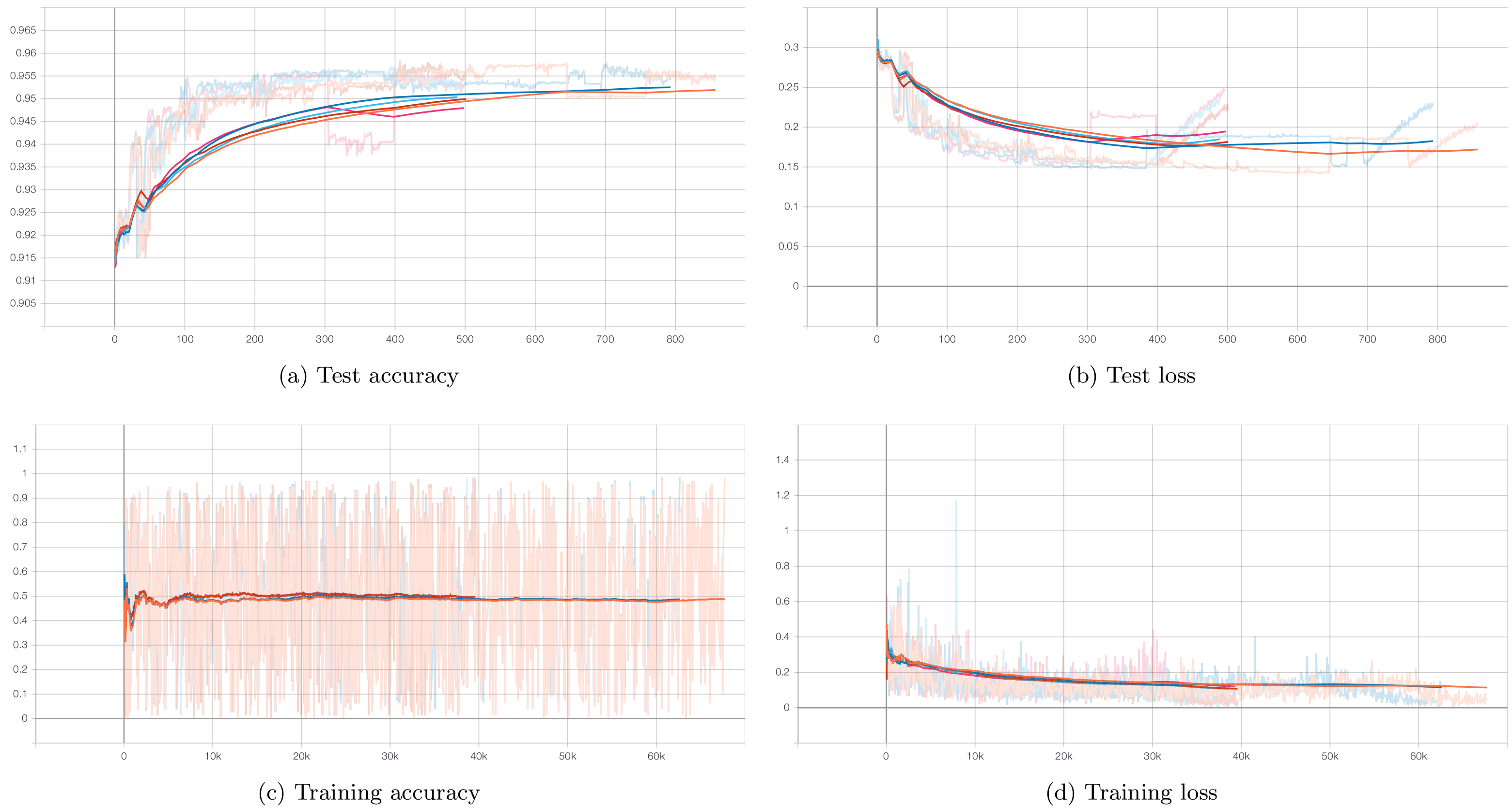


Figure B.2: Experimental results for ANTs with 20 growth steps per node and 100 global fine tune.

Comparison SDT to ANT

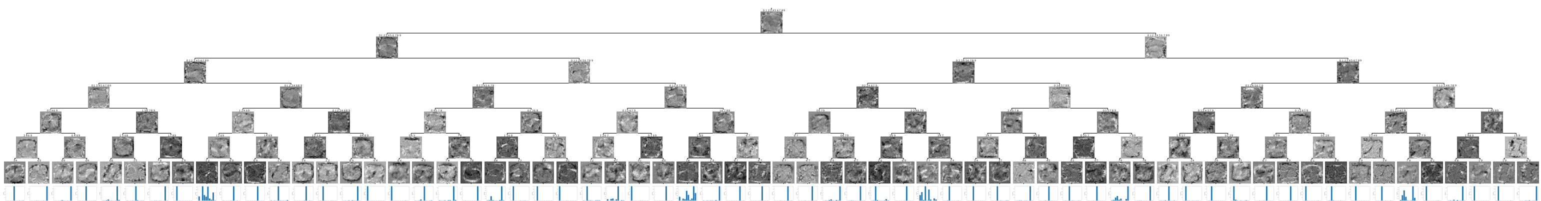


Figure B.3: SDT

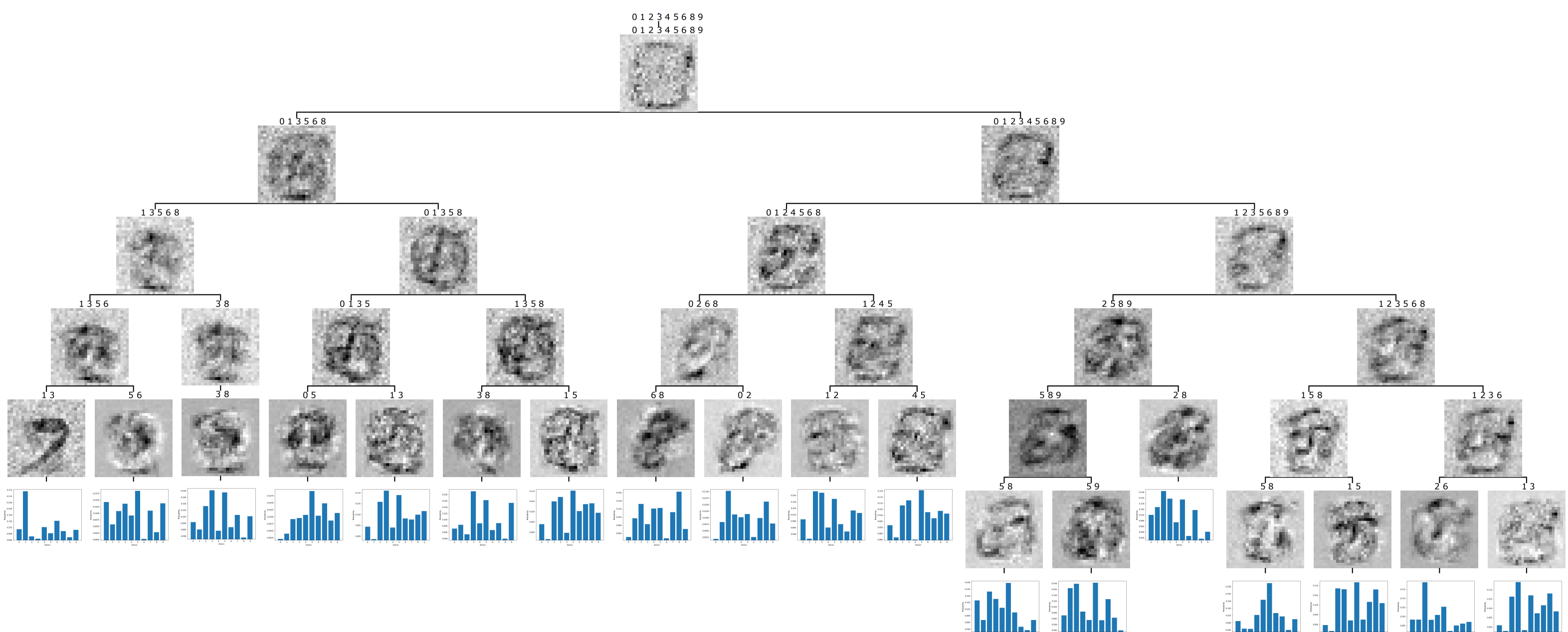


Figure B.4: ANT

Appendix C

OpenAI Gym

Ms Pacman

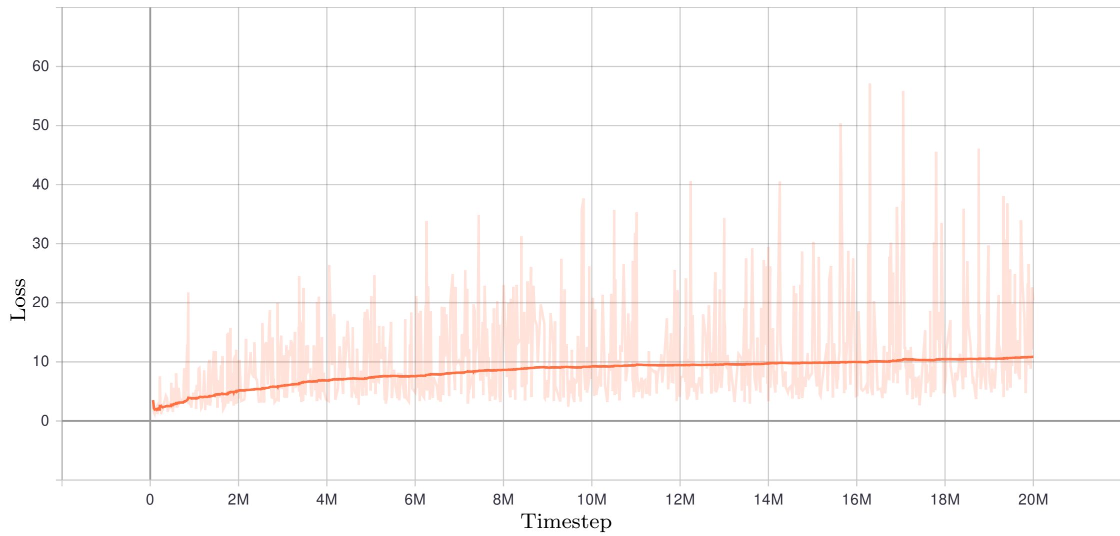


Figure C.1: Training loss of the best DQN agent.

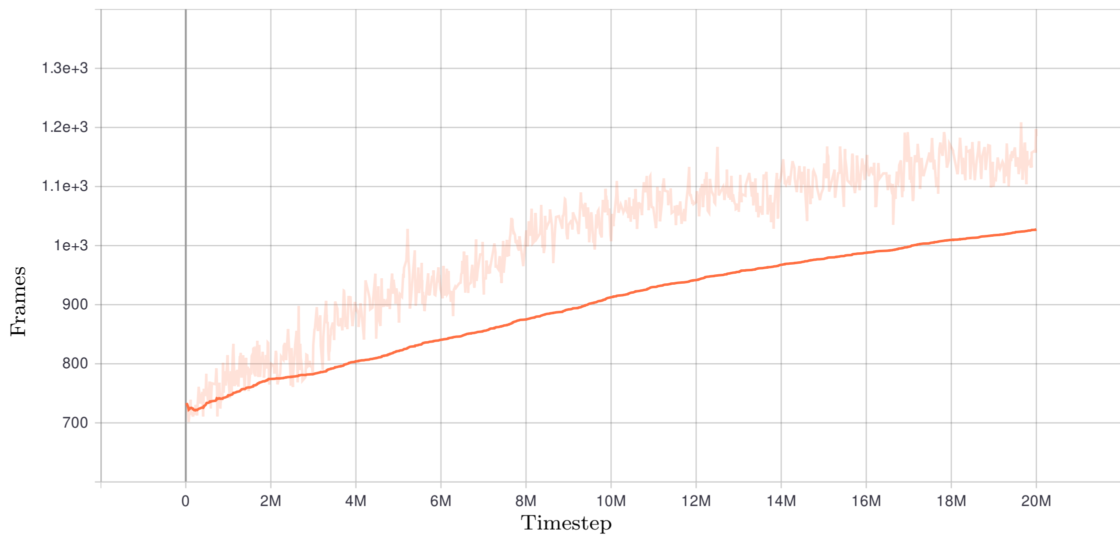


Figure C.2: Mean episode duration.

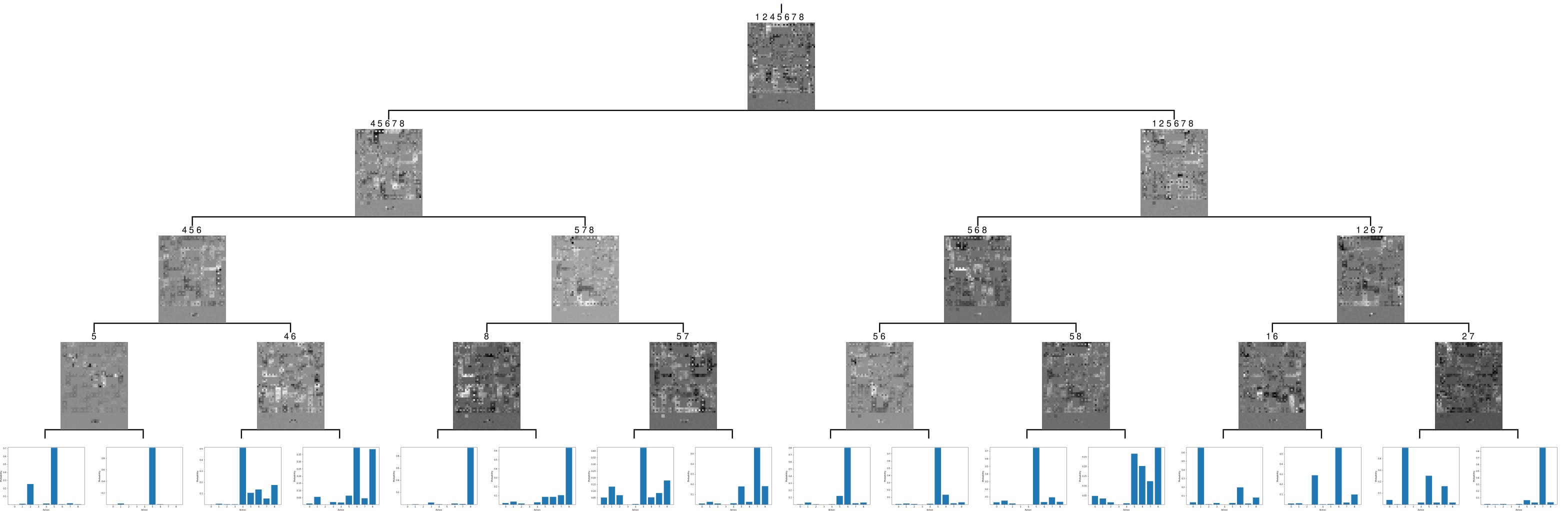


Figure C.3: Tree visualisation of SDT_{best}.

Enduro

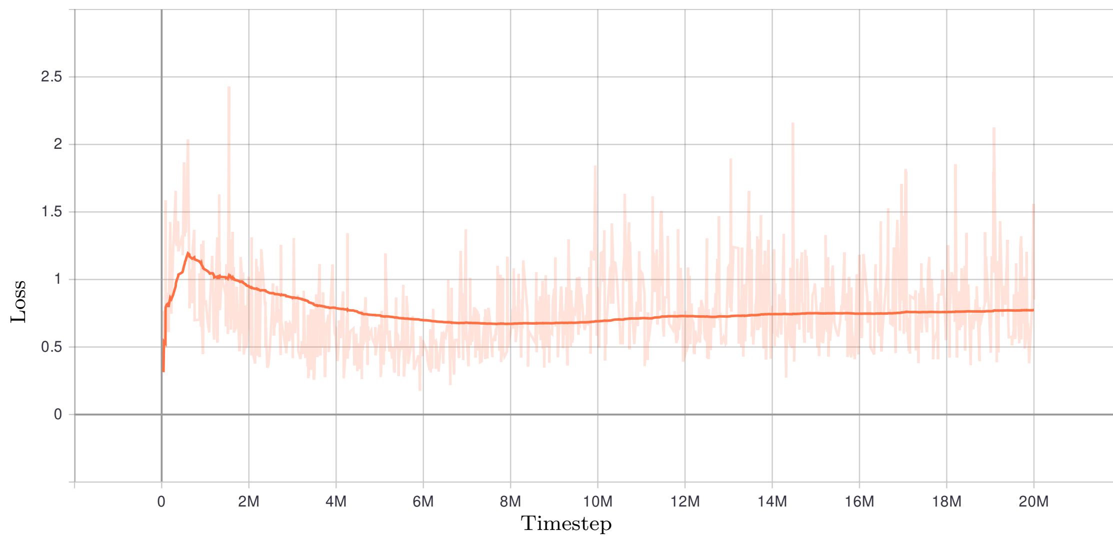


Figure C.4: Training loss of the best PPO agent.

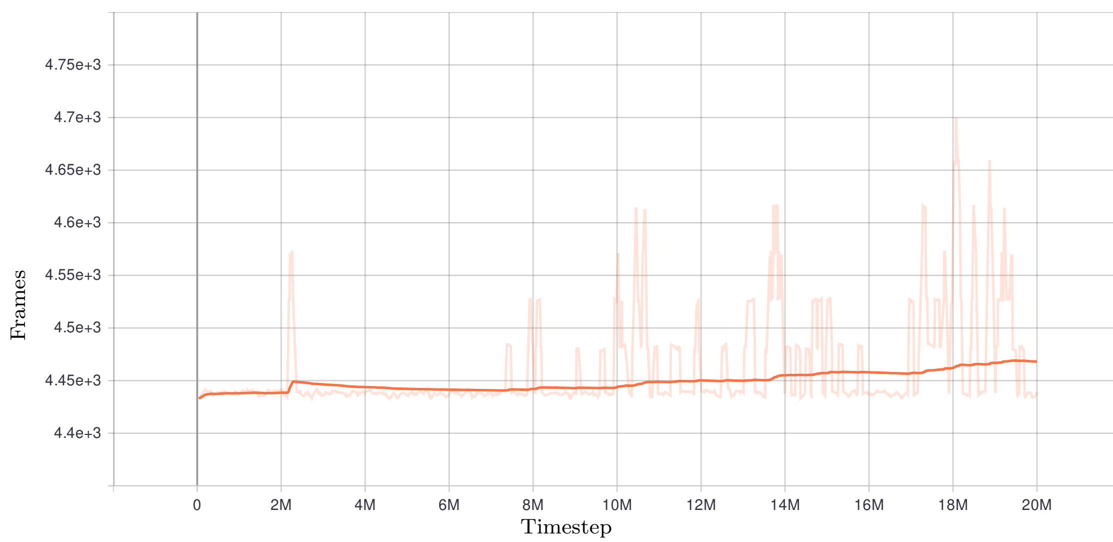


Figure C.5: Mean episode duration.

Figure C.6: Tree visualisation of SDT_{best}.

Bibliography

[1] H. Liao, J. Jiang, and Y. Zhang, "A study of automatic code generation," in *Proceedings - 2010 International Conference on Computational and Information Sciences, ICCIS 2010*, 2010, pp. 689–691.

[2] V. Y. Rosales-Morales, G. Alor-Hernández, J. L. García-Alcaráz, R. Zatarain-Cabada, and M. L. Barrón-Estrada, "An analysis of tools for automatic software development and automatic code generation," *Revista Facultad de Ingenieria*, vol. 2015, no. 77, pp. 75–87, 2015.

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

[4] T. M. Mitchell, *Machine Learning*, 1st ed. USA: McGraw-Hill, Inc., 1997.

[5] S. Nageshrao, H. E. Tseng, and D. Filev, "Autonomous Highway Driving using Deep Reinforcement Learning," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, oct 2019, pp. 2326–2331.

[6] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proceedings - IEEE International Conference on Robotics and Automation*. Singapore: IEEE, 2017, pp. 3389–3396.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, jan 2016.

[8] C. Molnar, *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.*, 2019. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[9] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, no. June 2017, pp. 1–38, feb 2017.

[10] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," pp. 206–215, may 2019.

[11] O. Loyola-Gonzalez, "Black-box vs. White-Box: Understanding their advantages and weaknesses from a practical point of view," pp. 154 096–154 113, 2019.

[12] B. Sohlberg and E. Jacobsen, "Grey box modelling branches and experiences," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 17, no. 1 PART 1, 2008.

[13] F. Doshi-Velez and B. Kim, "Towards A Rigorous Science of Interpretable Machine Learning," Tech. Rep., 2017. [Online]. Available: http://arxiv.org/abs/1702.08608

[14] Y. Coppens, K. Efthymiadis, T. Lenaerts, and A. Nowé, "Distilling Deep Reinforcement Learning Policies in Soft Decision Trees," in *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, 2019, pp. 1–6. [Online]. Available: https://cris.vub.be/files/46718934/IJCAI_2019_XAI_WS_paper.pdf

[15] O. T. Yildiz, E. Alpaydin, and O. Irsoy, "Soft decision trees," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, Tsukuba, Japan, 2012.

[16] S. Karakovskiy and J. Togelius, "The Mario Ai benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, mar 2012.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," Tech. Rep., mar 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[18] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori, "Adaptive neural trees," in *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, jul 2019, pp. 10 761–10 770. [Online]. Available: http://arxiv.org/abs/1807.06699

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," Tech. Rep., jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: The MIT Press Cambridge, 2018.

[21] E. L. Thorndike, *Animal Intelligence: Experimental Studies*, ser. Animal behavior series. Macmillan, 1911.

[22] R. Bellman, "The Theory of Dynamic Programming," p. 27, 1954.

[23] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, nov 2019. [Online]. Available: http://arxiv.org/abs/1808.00177

[24] S. Varges, G. Riccardi, S. Quarteroni, and A. V. Ivanov, "The exploration/exploitation trade-off in reinforcement learning for dialogue management," in *Proceedings of the 2009 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2009*. IEEE, 2009, pp. 479–484.

[25] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7006 LNAI, 2011, pp. 335–346.

[26] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 1st ed. USA: CRC Press, Inc., 2010.

[27] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev, "Why the Monte Carlo method is so important today," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, no. 6, pp. 386–392, nov 2014.

[28] H. Van Hasselt and M. A. Wiering, "Convergence of model-based temporal difference learning for control," in *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL 2007*, 2007, pp. 60–67.

[29] T. Jaakkola, M. L. Littman, C. Szepesvari, and S. Singh, "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms," *Machine Learning*, vol. 39, no. 1998, pp. 287–308, 2000.

[30] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms," *Neural Computation*, vol. 6, no. 6, pp. 1185–1201, 1994.

[31] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, may 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[32] M. Sewak, *Deep Reinforcement Learning - Frontiers of Artificial Intelligence.* Springer, 2019.

[33] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[34] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3607 LNAI, 2005, pp. 194–205.

[35] L. A. Zadeh, "Fuzzy Sets," *Information Control*, vol. 8, pp. 338–353, 1965.

[36] H. van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Adaptation, Learning, and Optimization*, 2012, vol. 12, pp. 207–251.

[37] G. Engeln-Müllges and F. Uhlig, *Linear and Nonlinear Approximation.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 179–218. [Online]. Available: https://doi.org/10.1007/978-3-642-61074-5_8

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015.

[39] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," Tech. Rep., dec 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15

[41] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010. [Online]. Available: www.ietdl.org

[42] E. Bonabeau, M. Dorigo, and G. Theraulaz, *From Natural to Artificial Swarm Intelligence.* USA: Oxford University Press, Inc., 1999.

[43] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to Humanoid Robotics.* Berlin: Springer Publishing Company, Incorporated, 2014. [Online]. Available: http://www.springer.com/series/5208

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* Cambridge, MA, USA: MIT press, 2016. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=ebf85b30d2d751196275d5dd14968935

[45] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, Eds., *Principles of Neural Science*, 3rd ed. New York: Elsevier, 1991.

[46] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[47] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Journal of Physiology*, vol. 117, pp. 500–544, 1952.

[48] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, ninth dove ed. New York: Dover, 1964.

[49] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," *Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018*, pp. 1836–1841, 2018.

[50] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Tech. Rep. 4, 1980.

[51] M. I. Jordan, *Serial order: A parallel distributed processing approach.*, ser. Advances in psychology, Vol. 121. Amsterdam, Netherlands: North-Holland/Elsevier Science Publishers, 1997.

[52] H. B. Curry, "The method of steepest descent for non-linear minimization problems," *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261, 1944.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: http://www.nature.com/articles/323533a0

[54] J. Martens, "Deep learning via Hessian-free optimization," in *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, 2010, pp. 735–742.

[55] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, 1990.

[56] C. R. Reeves, "Bias Estimation for Neural Network Predictions," in *Artificial Neural Nets and Genetic Algorithms.* Vienna: Springer Vienna, 1995, pp. 242–244.

[57] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[59] L. Prechelt, *Early Stopping — But When?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_5

[60] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A Study on Overfitting in Deep Reinforcement Learning," *CoRR*, vol. abs/1804.0, 2018. [Online]. Available: http://arxiv.org/abs/1804.06893

[61] K. Lee, K. Lee, J. Shin, and H. Lee, "Network Randomization: A Simple Technique for Generalization in Deep Reinforcement Learning," in *ICLR 2020*, 2020. [Online]. Available: http://arxiv.org/abs/1910.05396

[62] J. Tobin, R. H. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe. Vancouver, BC, Canada Domain: IEEE, 2017, pp. 23–30.

[63] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen, "Frame skip is a powerful parameter for learning to play atari," in *AAAI Workshop - Technical Report*, vol. WS-15-10, 2015, pp. 10–11. [Online]. Available: www.aaai.org

[64] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *ArXiv*, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[65] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405959519303455

[66] V. Mnih, A. Puigdomènech Badia, M. Mirza, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *International Conference on Machine Learning*, vol. 48, 2016. [Online]. Available: http://arxiv.org/abs/1301.3781

[67] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *ArXiv*, jul 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[68] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, feb 2015, pp. 1889–1897. [Online]. Available: http://arxiv.org/abs/1502.05477

[69] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[70] L. Weng, "Policy Gradient Algorithms," *lilianweng.github.io/lil-log*, 2018. [Online]. Available: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

[71] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, jun 2020.

[72] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*. Institute of Electrical and Electronics Engineers Inc., jan 2019, pp. 80–89.

[73] G. Dong, "Exploiting the Power of Group Differences: Using Patterns to Solve Data Analysis Problems," *Synthesis Lectures on Data Mining and Knowledge Discovery*, vol. 11, no. 1, pp. 1–146, feb 2019.

[74] Y. Ma and G. Guo, *Support Vector Machines Applications*. Springer Publishing Company, Incorporated, 2014.

[75] T. Bohlin, "Derivation of a designer's guide' for interactive grey-box' identification of nonlinear stochastic objects," *International Journal of Control*, vol. 59, no. 6, pp. 1505–1524, 1994.

[76] A. C. Scott, W. J. Clancey, R. Davis, and E. H. Shortliffe, "Explanation Capabilities of Production-Based Consultation Systems," *American Journal of Computational Linguistics*, pp. 1–50, feb 1977. [Online]. Available: https://www.aclweb.org/anthology/J77-1006

[77] W. R. Swartout, "Explaining and Justifying Expert Consulting Programs," in *Explaining and Justifying Expert Consulting Programs*, J. A. Reggia and S. Tuhrim, Eds. New York, NY, USA: Springer New York, 1985, vol. 2, ch. 29, pp. 254–271. [Online]. Available: https://doi.org/10.1007/978-1-4612-5108-8_15

[78] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu, "Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges," in *Natural Language Processing and Chinese Computing*, J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, Eds. Cham: Springer International Publishing, 2019, pp. 563–574.

[79] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[80] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[81] D. Michie, "Machine learning in the next five years," in *Proc. Third European Working Session on Learning*, ser. EWSL'88. USA: Pitman Publishing, Inc., 1988, pp. 107–122.

[82] W. R. Swartout and J. D. Moore, "Explanation in Second Generation Expert Systems," in *Second Generation Expert Systems*, J.-M. David, J.-P. Krivine, and R. Simmons, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 543–585.

[83] L. K. Hansen and L. Rieger, *Interpretability in Intelligent Systems A New Concept?*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, jan 2019, pp. 41–49.

[84] D. Gunning and D. W. Aha, "DARPA's explainable artificial intelligence program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, 2019.

[85] S. R. Haynes, M. A. Cohen, and F. E. Ritter, "Designs for explaining intelligent agents," *International Journal of Human Computer Studies*, vol. 67, no. 1, pp. 90–110, jan 2009.

[86] S. Anjomshoae, D. Calvaresi, A. Najjar, and K. Främling, "Explainable agents and robots: Results from a systematic literature review," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 2, 2019, pp. 1078–1088. [Online]. Available: www.ifaamas.org

[87] R. H. Wortham, A. Theodorou, and J. J. Bryson, "What Does the Robot Think? Transparency as a Fundamental Design Requirement for Intelligent Systems," in *IJCAI-2016 Ethics for Artificial Intelligence Workshop*, 2016. [Online]. Available: http://www.robwortham.com/instinct-planner/http://opus.bath.ac.uk/50294/1/WorthamTheodorouBryson{_}EFAI16.pdf

[88] N. Wang, D. V. Pynadath, and S. G. Hill, "The impact of POMDP-generated explanations on trust and performance in human-robot teams," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2016, pp. 997–1005. [Online]. Available: www.ifaamas.org

[89] S. Li, W. Sun, and T. Miller, "Communication in Human-Agent Teams for Tasks with Joint Action," in *Coordination, Organizations, Institutions, and Norms in Agent Systems XI*, V. Dignum, P. Noriega, M. Sensoy, and J. S. Sichman, Eds. Cham: Springer International Publishing, 2016, pp. 224–241.

[90] J. E. Colgate, W. Wannasuphoprasit, and M. A. Peshkin, "Cobots: robots for collaboration with human operators," in *American Society of Mechanical Engineers, Dynamic Systems and Control Division (Publication) DSC*, vol. 58, 1996, pp. 433–439.

[91] K. Baraka, A. Paiva, and M. Veloso, "Expressive Lights for Revealing Mobile Service Robot State," in *Robot 2015: Second Iberian Robotics Conference*, L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, Eds. Cham: Springer International Publishing, 2016, pp. 107–119.

[92] D. Holliday, S. Wilson, and S. Stumpf, "The Effect of Explanations on Perceived Control and Behaviors in Intelligent Systems," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 181–186. [Online]. Available: https://doi.org/10.1145/2468356.2468389

[93] F. Kaptein, J. Broekens, K. Hindriks, and M. Neerincx, "The role of emotion in self-explanations by cognitive agents," in *2017 7th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos, ACIIW 2017*, vol. 2018-Janua, 2018, pp. 88–93.

[94] M. Harbers, K. van den Bosch, and J.-J. C. Meyer, "A Study into Preferred Explanations of Virtual Agent Behavior," in *Intelligent Virtual Agents*, Z. Ruttkay, M. Kipp, A. Nijholt, and H. H. Vilhjálmsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 132–145.

[95] J. M. Alonso, "Explainable artificial intelligence for kids," in *Proceedings of the 11th Conference of the European Society for Fuzzy Logic and Technology, EUSFLAT 2019*, 2020, pp. 134–141. [Online]. Available: https://demos.citius.usc.es/ExpliClas/

[96] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, nov 2010. [Online]. Available: https://doi.org/10.1145/1868358.1868363

[97] K. V. Hindriks, "Debugging is explaining," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7455 LNAI, 2012, pp. 31–45.

[98] C. Rudin and B. Ustunb, "Optimized scoring systems: Toward trust in machine learning for healthcare and criminal justice," *Interfaces*, vol. 48, no. 5, pp. 449–466, 2018.

[99] I. Lana, J. J. Sanchez-Medina, E. I. Vlahogianni, and J. Del Ser, "From Data to Actions in Intelligent Transportation Systems: a Prescription of Functional Requirements for Model Actionability," *ArXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2002.02210

[100] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, "Big Data Analytics in Intelligent Transportation Systems: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 383–398, jan 2019.

[101] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, jan 2020.

[102] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, pp. 3319–3328.

[103] A. O. Afolabi and P. Toivanen, "Recommender systems in healthcare: Towards practical implementation of real-time recommendations to meet the needs of modern caregiving," in *Handbook of Research on Emerging Perspectives on Healthcare Information Systems and Informatics*, 2018, pp. 323–346.

[104] A. Adadi and M. Berrada, "Explainable AI for Healthcare: From Black Box to Interpretable Models," in *Embedded Systems and Artificial Intelligence*, V. Bhateja, S. C. Satapathy, and H. Satori, Eds. Singapore: Springer Singapore, 2020, pp. 327–337.

[105] M. R. Karim, M. Cochez, O. Beyan, S. Decker, and C. Lange, "OncoNetExplainer: Explainable predictions of cancer types based on gene expression data," *Proceedings - 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering, BIBE 2019*, pp. 415–422, 2019.

[106] A. Deeks, "The judicial demand for explainable artificial intelligence," *Columbia Law Review*, vol. 119, no. 7, pp. 1829–1850, 2019.

[107] P. Hacker, R. Krestel, S. Grundmann, and F. Naumann, "Explainable AI under contract and tort law: legal incentives and technical challenges," *Artificial Intelligence and Law*, 2020.

[108] T. Jiang, "Using Machine Learning to Analyze Merger Activity," Tech. Rep., 2018.

[109] K. Li, F. Mai, R. Shen, and X. Yan, "Corporate Culture and Merger Success: Evidence from Machine Learning," *UBC Sauder Working Paper*, 2018. [Online]. Available: https://editorialexpress.com/cgi-bin/conference/download.cgi?db_name=CICF2018&paper_id=394

[110] Council of European Union, "Council regulation (EU) no 269/2014," 2014.

[111] 2018 reform of EU data protection rules. [Online]. Available: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf

[112] L. Mitrou, "Is the General Data Protection Regulation (Gdpr) Artificial Intelligence-Proof," *SSRN*, no. December, 2018. [Online]. Available: https://ssrn.com/abstract=3386914

[113] P. Svenmarck, L. Luotsinen, M. Nilsson, and J. Schubert, "Possibilities and Challenges for Artificial Intelligence in Military Applications," *Proceedings of the NATO Big Data and Artificial Intelligence for Military Decision Making Specialists' Meeting*, pp. 1–17, 2018. [Online]. Available: https://www.researchgate.net/publication/326774966

[114] J. Su, D. V. Vargas, and K. Sakurai, "One Pixel Attack for Fooling Deep Neural Networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, oct 2019. [Online]. Available: http://arxiv.org/abs/1710.08864http://dx.doi.org/10.1109/TEVC.2019.2890858

[115] U. Pawar, D. O'Shea, S. Rea, and R. O'Reilly, "Explainable AI in Healthcare," in *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment, Cyber SA 2020*, 2020.

[116] M. A. Ahmad, A. Teredesai, and C. Eckert, "Interpretable machine learning in healthcare," in *Proceedings - 2018 IEEE International Conference on Healthcare Informatics, ICHI 2018*, 2018, p. 447.

[117] M. Van Lent, W. Fisher, and M. Mancuso, "An explainable artificial intelligence system for small-unit tactical behavior," in *Proceedings of the National Conference on Artificial Intelligence*, 2004, pp. 900–907. [Online]. Available: www.aaai.org

[118] M. van den Berg and O. Kuiper, "XAI in the Financial Sector. A Conceptual Framework for Explainable AI (XAI)," Tech. Rep., 2020. [Online]. Available: https://www-researchgate-net.ezproxy2.utwente.nl/publication/344079379_XAI_in_the_Financial_Sector_A_Conceptual_Framework_for_Explainable_AI_XAI%0Ahttps://www.hu.nl/-/media/hu/documenten/onderzoek/projecten/explainable_ai_in_the_financial_sector_van_den_b

[119] M. D. Fethi and F. Pasiouras, "Assessing bank efficiency and performance with operational research and artificial intelligence techniques: A survey," pp. 189–198, jul 2010.

[120] C. Chen, K. Lin, C. Rudin, Y. Shaposhnik, S. Wang, and T. Wang, "An Interpretable Model with Globally Consistent Explanations for Credit Risk," in *Proceedings of NeurIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the impact of Fairness, Explainability, Accuracy and privacy*, 2018. [Online]. Available: http://arxiv.org/abs/1811.12615

[121] K. Gade, S. Geyik, K. Kenthapadi, V. Mithal, and A. Taly, "Explainable AI in Industry: Practical Challenges and Lessons Learned," in *Companion Proceedings of the Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 303–304. [Online]. Available: https://doi.org/10.1145/3366424.3383110

[122] S. Luo, X. Lin, and Z. Zheng, "A novel CNN-DDPG based AI-trader: Performance and roles in business operations," *Transportation Research Part E: Logistics and Transportation Review*, vol. 131, pp. 68–79, nov 2019.

[123] A. Kusiak, "Artificial Intelligence Approach to Production Planning," in *Computer-Aided Production Management*, 1988, pp. 149–166.

[124] M. Fox, D. Long, and D. Magazzeni, "Explainable Planning," *ArXiv*, sep 2017. [Online]. Available: http://arxiv.org/abs/1709.10256

[125] F. K. Dosilovic, M. Brcic, and N. Hlupic, "Explainable artificial intelligence: A survey," *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pp. 210–215, 2018.

[126] A. A. Freitas, "Comprehensible Classification Models: A Position Paper," *SIGKDD Explor. Newsl.*, vol. 15, no. 1, pp. 1–10, mar 2014. [Online]. Available: https://doi.org/10.1145/2594473.2594475

[127] E. Puiutta and E. M. Veith, "Explainable Reinforcement Learning: A Survey," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12279 LNCS, may 2020, pp. 77–95. [Online]. Available: http://arxiv.org/abs/2005.06247

[128] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52 138–52 160, sep 2018.

[129] S. Sarkar, T. Weyde, A. D. Garcez, G. Slabaugh, S. Dragicevic, and C. Percy, "Accuracy and interpretability trade-offs in machine learning applied to safer gambling," in *CEUR Workshop Proceedings*, vol. 1773, 2016.

[130] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, no. 5, aug 2018.

[131] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, jan 2020.

[132] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," aug 2019.

[133] G. Vilone and L. Longo, "Explainable Artificial Intelligence: a Systematic Review," *ArXiv*, may 2020. [Online]. Available: http://arxiv.org/abs/2006.00093

[134] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[135] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, “Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation,” *Journal of Computational and Graphical Statistics*, vol. 24, no. 1, pp. 44–65, 2015. [Online]. Available: https://doi.org/10.1080/10618600.2014.907095

[136] D. W. Apley and J. Zhu, “Visualizing the effects of predictor variables in black box supervised learning models,” *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 82, no. 4, pp. 1059–1086, 2020.

[137] M. T. Ribeiro, S. Singh, and C. Guestrin, “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778

[138] T. Wang, C. Rudin, F. Velez-Doshi, Y. Liu, E. Klampfl, and P. MacNeille, “Bayesian Rule Sets for Interpretable Classification,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, dec 2016, pp. 1269–1274.

[139] X. Yin and J. Han, “CPAR: Classification based on Predictive Association Rules,” in *SDM*, 2003.

[140] D. M. Malioutov, K. R. Varshney, A. Emad, and S. Dash, “Learning Interpretable Classification Rules with Boolean Compressed Sensing,” *Transparent Data Mining for Big and Small Data*, pp. 95–121, 2017.

[141] H. Lakkaraju, S. H. Bach, and J. Leskovec, “Interpretable Decision Sets: A Joint Framework for Description and Prediction,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1675–1684. [Online]. Available: https://doi.org/10.1145/2939672.2939874

[142] J. Krause, A. Perer, and K. Ng, *Interacting with Predictions: Visual Inspection of Black-Box Machine Learning Models*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 5686–5697. [Online]. Available: https://doi.org/10.1145/2858036.2858529

[143] P. Adler, C. Falk, S. A. Friedler, T. Nix, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian, “Auditing Black-Box Models for Indirect Influence,” *Knowl. Inf. Syst.*, vol. 54, no. 1, pp. 95–122, jan 2018. [Online]. Available: https://doi.org/10.1007/s10115-017-1116-3

[144] J. J. Thiagarajan, B. Kailkhura, P. Sattigeri, and K. N. Ramamurthy, “TreeView: Peeking into Deep Neural Networks Via Feature-Space Partitioning,” 2016. [Online]. Available: http://arxiv.org/abs/1611.07429

[145] J. Adebayo and L. Kagal, “Iterative Orthogonal Feature Projection for Diagnosing Bias in Black-Box Models,” *CoRR*, vol. abs/1611.0, 2016. [Online]. Available: http://arxiv.org/abs/1611.04967

[146] J. D. Olden and D. A. Jackson, “Illuminating the black box: a randomization approach for understanding variable contributions in artificial neural networks,” *Ecological Modelling*, vol. 154, no. 1, pp. 135–150, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304380002000649

[147] H. A. Chipman, E. I. George, and R. E. Mcculloch, *Making sense of a forest of trees 1 Introduction 2 Methods for generating trees*, 1998.

[148] M. Craven and J. W. Shavlik, "Using Sampling and Queries to Extract Rules from Trained Neural Networks," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ser. ICML'94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 37–45.

[149] A. Henelius, K. Puolamaki, H. Bostrom, L. Asker, and P. Papapetrou, "A peek into the black box: exploring classifiers by randomization," *Data Mining and Knowledge Discovery*, vol. 28, no. 5-6, pp. 1503–1529, sep 2014.

[150] M. W. Craven and J. W. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, ser. NIPS'95. Cambridge, MA, USA: MIT Press, 1995, pp. 24–30.

[151] H. F. Tan, G. Hooker, and M. Wells, "Tree Space Prototypes: Another Look at Making Tree Ensembles Interpretable," *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, 2020.

[152] R. Turner, "A model explanation system," in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2016, pp. 1–6.

[153] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2016, pp. 2921–2929.

[154] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, oct 2017, pp. 618–626.

[155] C. E. Shannon, "A mathematical theory of communication." *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948. [Online]. Available: http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48

[156] F. Girosi, M. Jones, and T. Poggio, "Regularization Theory and Neural Networks Architectures," *Neural Computation*, vol. 7, no. 2, pp. 219–269, 1995. [Online]. Available: https://doi.org/10.1162/neco.1995.7.2.219

[157] N. Patel and S. Upadhyay, "Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA," *International Journal of Computer Applications*, vol. 60, no. 12, pp. 20–25, 2012.

[158] J. Mingers, "Rule induction with statistical dataa comparison with multiple regression," *Journal of the Operational Research Society*, vol. 38, no. 4, pp. 247–251, 1987. [Online]. Available: www.jstor.org

[159] S. Klinke and J. Grassmann, "Visualization and Implementation of Feedforward Neural Networks," Tech. Rep., 1996.

[160] G. Kogan, "ml4a: Looking inside neural nets," 2020. [Online]. Available: https://ml4a.github.io/ml4a/looking_inside_neural_nets/

[161] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8689 LNCS, no. PART 1, 2014, pp. 818–833.

[162] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8818. Springer Verlag, 2014, pp. 364–375.

[163] J. Fürnkranz, D. Gamberger, and N. Lavrac, *Foundations of Rule Learning*, ser. Cognitive Technologies. Springer, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75197-7

[164] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, no. 4, pp. 261–283, 1989. [Online]. Available: https://doi.org/10.1007/BF00116835

[165] W. W. Cohen, "Fast Effective Rule Induction," in *In Proceedings of the Twelfth International Conference on Machine Learning.* Morgan Kaufmann, 1995, pp. 115–123.

[166] J. H. Friedman and N. I. Fisher, "Bump hunting in high-dimensional data," *Statistics and Computing*, vol. 9, no. 2, pp. 123–143, 1999.

[167] Y. Coppens, D. Steckelmacher, C. M. Jonker, and A. Nowé, "Synthesising Reinforcement Learning Policies through Set-Valued Inductive Rule Learning," 2020.

[168] T. G. Dietterich, "Ensemble Methods in Machine Learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. London, UK: Springer-Verlag, 2000, pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=648054.743935

[169] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, 1st ed. Chapman & Hall/CRC, 2012.

[170] S. J. Hanson and L. Y. Pratt, "Comparing Biases for Minimal Network Construction with Back-Propagation," *Advances in Neural Information Processing Systems (NIPS)*, vol. 1, pp. 177–185, 1989. [Online]. Available: http://portal.acm.org/citation.cfm?id=89851.89872

[171] C. Bucil, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 2006, 2006, pp. 535–541.

[172] D. Lowd and P. Domingos, "Naive Bayes models for probability estimation," in *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, 2005, pp. 529–536.

[173] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Y. LeCun and Y. Bengio, Eds., San Diego, CA, USA, may 2015. [Online]. Available: http://arxiv.org/abs/1412.6550

[174] G. K. Nayak, K. R. Mopuri, V. Shaj, R. Venkatesh Babu, and A. Chakraborty, "Zero-shot knowledge distillation in deep networks," in *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, 2019, pp. 8317–8325.

[175] N. Frosst and G. Hinton, "Distilling a Neural Network Into a Soft Decision Tree," Tech. Rep., nov 2017. [Online]. Available: http://arxiv.org/abs/1711.09784

[176] M. I. Jordan and R. A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," *Neural Computing*, vol. 6, no. 2, pp. 181–214, mar 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.2.181

[177] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[178] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, jul 2017.

[179] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *arXiv*, aug 2020. [Online]. Available: http://arxiv.org/abs/2008.06693

[180] T. Lesort, N. Díaz-Rodríguez, J. F. Goudou, and D. Filliat, "State representation learning for control: An overview," pp. 379–392, feb 2018. [Online]. Available: http://arxiv.org/abs/1802.04181http://dx.doi.org/10.1016/j.neunet.2018.07.006

[181] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable Reinforcement Learning via Reward Decomposition," *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, pp. 47—-53, 2019.

[182] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid Reward Architecture for Reinforcement Learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5392–5402. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/1264a061d82a2edae1574b07249800d6-Paper.pdf

[183] A. Alharin, T.-N. Doan, and M. Sartipi, "Reinforcement Learning Interpretation Methods: A Survey," *IEEE Access*, vol. 8, pp. 171 058–171 077, sep 2020.

[184] Z. Zhang and X. Ji, "Regret minimization for reinforcement learning by evaluating the optimal bias function," 2019.

[185] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 1563–1600, 2010.

[186] J. Insa-Cabrera, D. L. Dowe, and J. Hernández-Orallo, "Evaluating a reinforcement learning algorithm with a general intelligence test," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7023 LNAI, 2011, pp. 1–11.

[187] J. Hernández-Orallo and D. L. Dowe, "Measuring universal intelligence: Towards an anytime intelligence test," *Artificial Intelligence*, vol. 174, no. 18, pp. 1508–1539, dec 2010.

[188] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, may 2017. [Online]. Available: https://doi.org/10.1145/3065386

[189] S. Lange and M. A. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning." in *IJCNN*. IEEE, 2010, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2010.html#LangeR10

[190] M. Riedmiller, "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method," in *Machine Learning: ECML 2005*, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328.

[191] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2009, pp. 2146–2153.

[192] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines." in *ICML*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2010.html{#}NairH10

[193] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable Baselines 3," https://github.com/DLR-RM/stable-baselines3, 2019.

[194] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "OpenAI Baselines," https://github.com/openai/baselines, 2017.

[195] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *ArXiv*, nov 2016. [Online]. Available: http://arxiv.org/abs/1611.05763

[196] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like Environment for Machine Learning," in *BigLearn, NIPS Workshop*, 2011.

[197] M. Mitchell, *Complexity: A Guided Tour.* Oxford University Press, USA, 2009.

[198] M. Lima, *Visual Complexity: Mapping Patterns of Information.* Princeton Architectural Press, 2013.

[199] H. Buhrman and R. De Wolf, "Complexity measures and decision tree complexity: A survey," in *Theoretical Computer Science*, vol. 288, no. 1, 2002, pp. 21–43. [Online]. Available: www.elsevier.com/locate/tcs

[200] I. Chikalov, "Bounds on Average Time Complexity of Decision Trees," *Intelligent Systems Reference Library*, vol. 21, pp. 15–39, 2011.

[201] J. Hartmanis and R. E. Stearns, "On the Computational Complexity of Algorithms," *Journal of Symbolic Logic*, vol. 32, no. 1, pp. 120–121, 1967.

[202] P. Bossaerts and C. Murawski, "Computational Complexity and Human Decision-Making," pp. 917–929, dec 2017.

[203] D. Peebles and R. P. Cooper, "Thirty years after Marr's vision: Levels of analysis in cognitive science," *Topics in Cognitive Science*, vol. 7, no. 2, pp. 187–190, apr 2015.

[204] D. Marr, *Vision: a computational investigation into the human representation and processing of visual information.* London, UK: The MIT Press, 1982.

[205] M. Pantsar, "Cognitive and Computational Complexity: Considerations from Mathematical Problem Solving," *Erkenntnis*, 2019.

[206] J. K. Tsotsos, "How does human vision beat the computational complexity of visual perception," in *Computational processes in human vision: an interdisciplinary perspective*, 1988, pp. 286–338.

[207] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines," https://github.com/hill-a/stable-baselines, 2018.

[208] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and Others, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[209] J. Huerta-Cepas, F. Serra, and P. Bork, "ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data," *Molecular Biology and Evolution*, vol. 33, no. 6, pp. 1635–1638, jun 2016.

[210] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[211] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[212] M. Grinberg, *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[213] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *J. Artif. Int. Res.*, vol. 47, no. 1, pp. 253–279, may 2013.

[214] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[215] A. Byerly, T. Kalganova, and I. Dear, "A Branching and Merging Convolutional Network with Homogeneous Filter Capsules," *ArXiv*, vol. abs/2001.0, 2020.

[216] P. Simard, Y. Le Cun, and J. Denker, "Efficient Pattern Recognition Using a New Transformation Distance," in *Advances in Neural Information Processing Systems 5*, 1992, pp. 51–58.