Rule extraction from deep neural networks

Master’s Thesis

Adviser: Assistant Professor Aleksander Sadikov
Coadviser: Professor Mateja Jamnik
Coadviser: Dr. Zohreh Shams
Urška Matjašec

EKSTRAKCIJA PRAVIL IZ GLOBOKIH NEVRONSKIH MREŽ

Magistrsko delo

Mentor: doc. dr. Aleksander Sadikov
Somentorica: prof. dr. Mateja Jamnik
Somentorica: znan. sod. dr. Zohreh Shams

Ljubljana, 2020
Acknowledgements

First, I would like to thank my thesis mentor, Prof. Aleksander Sadikov, for his guidance and help with the thesis.

I am extremely grateful to Prof. Mateja Jamnik and Dr. Zohreh Shams, without whom this work would not exist. Thank you for your knowledge and guidance and for introducing me to the world of research.

Finally, I would like to express my profound gratitude to Jernej and my family for their encouragement, support, and belief in me throughout my studies.
# Contents

Project plan xi

1 Introduction 1
   1.1 Contributions .................................................. 3
   1.2 Thesis structure ................................................ 3

2 Background and literature review 5
   2.1 Clinical decision support systems ............................ 5
   2.2 Deep learning .................................................... 6
   2.3 Deep feedforward neural networks ............................ 6
      2.3.1 Definitions .................................................. 9
   2.4 Approaches to rule extraction ................................. 10
   2.5 IF-THEN rules ................................................... 11
   2.6 CRED .............................................................. 13
   2.7 DeepRED ........................................................... 15
   2.8 C4.5 ................................................................. 18
   2.9 C5.0 ................................................................. 20
   2.10 Summary .......................................................... 21

3 Design and implementation 23
   3.1 Design ............................................................ 23
   3.2 Rule extraction algorithm ....................................... 26
      3.2.1 Building the rules .......................................... 27
      3.2.2 Merging the rule sets ...................................... 30
   3.3 Data ............................................................... 34
   3.4 Experimental setup .............................................. 35
   3.5 Evaluation measures ............................................ 37

4 Experiments and evaluation 41
   4.1 Mapping mRNA to ER expressions ............................. 43
   4.2 Comparison of our algorithm and DeepRED .................... 45

5 Analysis 47
   5.1 Theoretical complexity .......................................... 47
      5.1.1 Building the rules .......................................... 47
      5.1.2 Merging the rule sets ...................................... 50
   5.2 Comparison with two baselines ................................. 51

6 Conclusion 59
   6.1 Summary .......................................................... 59
   6.2 Future work ........................................................ 60

7 Razširjeni povzetek v slovenščini 61
   7.1 Motivacija .......................................................... 61
   7.2 Cilji ................................................................. 61
   7.3 Globoke usmerjene nevronske mreže ........................... 62
7.4 Načrt za evaluacijo algoritma ........................................ 63
7.5 Algoritem za ekstrakcijo pravil ....................................... 66
   7.5.1 Gradnja pravil .................................................. 67
   7.5.2 Združevanje pravil ............................................. 68
7.6 Rezultati ............................................................... 68
7.7 Zaključek .............................................................. 71

A Additional results for our rule extraction algorithm and the pedagogical C5.0 algorithm ........................................ 73
   A.1 Mapping mRNA and CNA to ER expressions ............... 73
   A.2 Mapping mRNA and CNA to the PAM50 feature ............ 74

References ........................................................................ 77
List of Figures

1. General structure of a deep feedforward neural network. ........................................ 7
2. A simple mathematical model of a neuron [37]. .................................................... 8
3. Pseudo-code of the back-propagation algorithm [37]. .............................................. 9
4. The CRED algorithm (adapted from [40]). .............................................................. 14
5. Pseudo-code of the DeepRED algorithm [53]. .......................................................... 16
6. Decision tree built with C4.5. .................................................................................. 19
7. The entire pipeline for evaluating our rule extraction algorithm. .............................. 24
8. The architecture of networks for the three experiments. \( DNN_1 \) has 1,000
   neurons in the input layer and two neurons in the output layer; \( DNN_2 \)
   has 2,000 input neurons and two output neurons; \( DNN_3 \) has 2,000 inputs
   and five output neurons. .......................................................................................... 25
9. Decision tree, mapping the second hidden layer to the output layer. ....................... 28
10. Decision trees, mapping the first hidden layer to the second hidden layer. ............... 29
11. Decision trees, mapping the input layer to the first hidden layer. ............................ 30
12. Decision tree, mapping the first hidden layer to the output layer. ........................... 31
13. Decision tree, mapping the input layer to the output layer. ..................................... 33
14. Rule sets for two consecutive layers. ..................................................................... 48
15. Pseudo-code for the \( i \)-th merge step. ...................................................................... 50
16. Comparing time and memory of the three algorithms for model \( DNN_1 \) on
   a logarithmic scale. ................................................................................................. 51
17. Comparing the accuracy and fidelity of rules that are extracted with the
   three algorithms for model \( DNN_1 \). ........................................................................ 52
18. Comparing rule set size and average rule length of rules that are extracted
   with the three algorithms for model \( DNN_1 \). ......................................................... 53
19. Comparing rule set size and accuracy of rules that are extracted with the
   three algorithms for model \( DNN_1 \) (horizontal axis is displayed on a logarithmic scale).
   .............................................................................................................................. 53
20. Comparing average rule length and accuracy of rules that are extracted with
    the three algorithms for model \( DNN_1 \). ................................................................. 54
21. Comparing rule set size and fidelity of rules that are extracted with the
    three algorithms for model \( DNN_1 \) (horizontal axis is displayed on a logarithmic scale).
    .............................................................................................................................. 55
22. Comparing average rule length and fidelity of rules that are extracted with
    the three algorithms for model \( DNN_1 \). ................................................................. 55
23. Number of features that appear in both rule sets, generated with our
    algorithm and DeepRED, in all five folds or in a subset of folds (for model
    \( DNN_1 \)). .................................................................................................................. 56
24. Venn diagram, representing features that appear in all three rule sets
    (generated by our algorithm and the two baselines) across five folds (for
    model \( DNN_1 \)). ........................................................................................................ 57
25. Matematični model nevrona [37]. ........................................................................ 63
Arhitektura nevronskih mrež za tri probleme. $DNN_1$ ima 1.000 nevronov v vhodnem nivoju in dva nevrona v izhodnem nivoju; $DNN_2$ ima 2.000 nevronov v vhodnem nivoju in dva nevrona v izhodnem nivoju; $DNN_3$ ima 2.000 nevronov v vhodnem nivoju in pet nevronov v izhodnem nivoju.  

Primerjava časovne in prostorske komponente treh algoritmov (pedagoški algoritem C5.0, naš algoritem in DeepRED) za model $DNN_1$.  

Primerjava napovedne točnosti in zvestobe nevronski mreži pravil, zgradjenih s trema algoritmi za model $DNN_1$.  

Primerjava števila izvlečenih pravil in povprečne dolžine pravil, zgradjenih s trema algoritmi za model $DNN_1$.  

Vennov diagram, ki prikazuje najbolj pogoste atribute v petih različnih delih, ki se pojavijo v vseh množicah, ki so zgrajene z našim algoritmom in obema osnovnicama (za model $DNN_1$).
Program dela

1. Poišči najboljše parametre za globoko nevronsko mrežo z dvema skritima nivojema, 1.000 vhodnimi atributi in dvema izhodnima nevronoma.

2. Izračunaj povprečno napovedno točnost na testni množici s 5-kratnim prečnim preverjanjem.

3. Uporabi predstavljeni algoritem za ekstrakcijo pravil na izbrani nevronski mreži in oceni vrnjena pravila. Uporabi sledeča algoritma za ekstrakcijo pravil in oceni zgrajeni množici pravil za primerjavo z našim algoritmom:
   - pedagoški C5.0 in
   - DeepRED.

4. Povečaj število vhodnih atributov in ponovi korake 1-3 za globoko nevronsko mrežo z dvema skritima nivojema, 2.000 vhodnimi atributi in dvema izhodnima nevronoma.

5. Povečaj število izhodnih nevronov (večcilnja klasifikacija) in ponovi korake 1-3 za globoko nevronsko mrežo z dvema skritima nivojema, 2.000 vhodnimi atributi in petimi izhodnimi nevroni.

6. Analiziraj kompleksnost predstavljenega algoritma.
**Work plan**

1. Perform a parameter sweep for a simple deep neural network with two hidden layers: when the size of the input layer is 1,000 and the size of the output layer is two.

2. Calculate the average accuracy of the test set over 5-fold cross validation.

3. Use our algorithm to extract rules from the best neural network. Evaluate the extracted rules and compare them with the extracted rules using the following algorithms:
   - pedagogical C5.0 and
   - DeepRED.

4. Repeat 1, 2, and 3 for more input features: when the size of the input layer is 2,000 and the size of the output layer is two.

5. Repeat 1, 2, and 3 for multi-class classification: when the size of the input layer is 2,000 and the size of the output layer is five.

6. Analyse the complexity of our algorithm.
Osnovna literatura (Basic references)


Podpis mentorja (adviser):

Podpis so mentorja (coadviser):

Podpis so mentorja (coadviser):
Ekstrakcija pravil iz globokih nevronskih mrež

POVZETEK

Kljub visoki napovedni točnosti imajo globoke nevronske mreže veliko pomanjkljivost, ki jim preprečuje široko uporabo v varnostnih in varnostno kritičnih sistemih, na primer v zdravstvu. Ta ovira je nezmožnost pojasnjevanja njihovih odločitev. V namen interpretacije nevronskih mrež z ekstrakcijo pravil je bilo razvitih že kar nekaj algoritmov, vendar imajo vsi vsaj eno izmed sledečih pomanjkljivostí: ignorirajo skrite nivoje nevronsko mreže in tako zanemario informacije iz notranjosti mreže, niso uporabni na globokih nevronskih mrežah, ali pa so časovno in prostorsko preveč zahtevni.

To magistrsko delo predstavi razširljiv, dekompozični algoritem za ekstrakcijo preprostih in razumljivih če-poštet pravil iz globokih nevronskih mrež, ki je sposoben reševanja problemov veččilne klasifikacije. Izveščena pravila aproksimirajo obnašanje nevronske mreže in razlagajo izhodni nivo mreže z vhodnimi atributami. Algoritem smo uporabili na zdravstveni domeni, kjer je razložljivost ključna, in ga testirali na podatkovni bazi METABRIC, ki zajema podatke o pacientkah z rakom dojk. Algoritem smo uporabili za reševanje problemov binarne in veččilne klasifikacije in ga primerjali z dvema osnovnicama: pedagoškim algoritmom C5.0 in dekompozičnim algoritmom DeepRED.

Po pričakovanjih se je pedagoški algoritem C5.0 izkazal najbolje v časovni in prostorski zahtevnosti, številu izveščenih pravil in njihovi povprečni dolžini. Vendar pa je naš algoritem zgradil pravila z boljšo napovedno točnostjo in večjo zvestobo nevronski mreži kot pedagoška osnovnica. Naš algoritem je v vseh pogledih tudi prekosi dekompozičnem algoritmu DeepRED.


Ključne besede: razložljivost nevronskih mrež, globoke usmerjene nevronke mreže, ekstrakcija pravila, odločilvena pravila, dekompozični algoritem
Rule extraction from deep neural networks

Abstract

Despite their high accuracy, deep neural networks' main disadvantage is their lack of transparency, interpretability and explainability. This has prevented them from being widely used in safety and security critical systems, for example, healthcare. There have been many attempts to interpret neural networks via rule extraction. However, the existing algorithms either do not take into account the entire structure of the network, are not applicable on deep neural networks, or are very time and memory expensive.

This thesis introduces a scalable, decompositional rule extraction algorithm, which can extract simple, easy to understand IF-THEN rules from deep neural networks and can deal with the multi-class classification problems. The extracted rules approximate the network's behaviour and explain its output layer in terms of the input features. The algorithm was applied in the healthcare domain, where explainability is crucial. It was tested on a data set of breast cancer patients (METABRIC), and evaluated on two tasks of binary and multi-class classification problems.

The algorithm's performance was compared to two baselines: pedagogical C5.0 and the decompositional algorithm DeepRED. As expected, the pedagogical baseline outperformed both decompositional algorithms in time and memory complexity, number of extracted rules and their average length. However, our algorithm provided more accurate rules with a higher level of fidelity than the pedagogical baseline. Our algorithm also outperformed the decompositional baseline DeepRED in all perspectives.


Keywords: explainability of neural networks, deep feedforward neural networks, rule extraction, decision rules, decompositional algorithm

xix
1 Introduction

Deep learning, an approach in machine learning, has achieved many notable successes in recent years. Deep learning methods use neural networks, which is why they are often referred to as deep neural networks. These are complex neural networks with more than one hidden layer. They are based on biological neural networks and are capable of identifying complex nonlinear relationships between input and output data. Because they are very accurate for making predictions, we use them for a variety of applications, for example, automated driving, industrial automation, electronics, natural language processing, and medical research.

Despite their high accuracy, deep neural networks' main disadvantage is their lack of transparency, interpretability and explainability. This has prevented them from being widely used in safety and security critical systems, for example, healthcare. Neural networks are not able to provide explanations for their decisions, which is a problem in applications, where it is crucial for us to understand the decision and reasons that lead to it, for example, when using deep neural network for predicting if a patient's cancer will reoccur.

There have been many attempts to interpret neural networks. Fürnkranz et al. [14] argue that rules provide the best trade-off between human and machine understandability. For that reason, we explore rule extraction as a way to explain neural networks. There exist three approaches for rule extraction from neural networks: pedagogical, decompositional, and eclectic [2]. The pedagogical approach ignores neural network's structure and uses only data from the input and output layers. Decompositional approach, on the other hand, takes into account the whole architecture - input, output, and also the network's internal structure represented by hidden layers. Algorithms of the eclectic approach have some features of both of these approaches. All approaches approximate deep neural networks, thus trade-off some accuracy in order to gain interpretability.

Both pedagogical and decompositional approaches have some benefits and disadvantages. Regarding computational complexity, pedagogical algorithms are simpler, because they take into consideration only input and output layers of the neural network. Thus, they forego a big part of the information from the network's inner structure. Decompositional algorithms, on the other hand, split the network into neuron levels and aggregate the results obtained from each neuron to represent the neural network as a whole. This requires large time and memory resources, which increases significantly with greater number of hidden layers. All things considered, if we could better manage time and memory complexity, decompositional algorithms better explain neural networks, because they take their whole structure into consideration.

There are a number of decompositional approaches to explaining neural network's decisions in terms of the input features. The one we are interested in uses simple IF-THEN rules. In [52], authors introduce the algorithm DeepRED, which extracts IF-THEN rules from deep neural networks, which are structured in very simple and easily understandable decision trees. They do so not directly on data, but on the neural networks, trained on the data. Their algorithm looks at the activation values on all layers and produces rules that predict the output depending on the features from the input layer. In a sense, they model neural network's behaviour with decision trees and simple
rules. This was the first attempt to extract such rules from deep neural networks. Previous algorithms were all devised for shallow networks, that is, neural networks with one hidden layer.

DeepRED experiments were all done on binary classification problems. Authors report some problems with time and memory [52], that is, a few experiments exceeded allocated memory space (10,000 MB) or maximum execution time (24 h). Although DeepRED brings some novelty to the rule extraction field, it suffers from scalability.

Inspired by DeepRED, we propose a decompositional algorithm for deep neural networks that is time and memory efficient, and also more scalable. In the first part, our algorithm extracts decision trees for two adjacent layers of the neural network, starting at the output layer and making its way towards the input layer. In each step, it generates a set of rules, which map from one layer to the following layer. In the second part, the algorithm merges two sets of rules, again layer by layer, going from the output towards the input layer. In each step, we get a set of rules that map from one layer before to the output layer. When the merging is done, we get a set of rules that map input layer features, to the output layer predictions.

Given that DeepRED is the only algorithm that extracts rules from deep neural networks, devising an algorithm that outperforms DeepRED makes our algorithm the most efficient decompositional approach to date, to the best of our knowledge. We evaluate the performance of our extracted rules by measuring accuracy, length of rules and the size of the rule set, and compare them to the two baselines, where higher accuracy, shorter rules and smaller rule sets are favoured. The extracted rule sets mimic the behaviour of the neural network, so we also include fidelity as one of the measures. Fidelity compares the predictions of a rule set with the predictions of a corresponding neural network. In addition, we theoretically present algorithm's complexity analysis.

The first baseline is DeepRED algorithm, which uses a decompositional approach because it looks at the information from all layers. One of the main steps in DeepRED is generating decision trees with the classification algorithm C4.5. We improve it by using C4.5's successor, the classification algorithm C5.0, for generating IF-THEN rules. We achieve improvements in accuracy, the size of the final rule set and the average rule length. We want the rules explaining neural network to be compact and easy to understand.

The second baseline is pedagogical C5.0, that is, using the classification algorithm C5.0 on the raw data and the output of the neural network. This way, we ignore network's hidden layers, which makes this a pedagogical approach to extracting rules. In general, decompositional approach is better because it also considers the internal structure of a network, rather than just using information from the input and output layers. But, as a consequence, it is also computationally more expensive. We want to find out how this cost scales compared to the pedagogical approach.

We apply our algorithm in the healthcare domain, where explainability is crucial. For testing our algorithm, we use the METABRIC data set, which consists of 1,980 breast-cancer patients. METABRIC (Molecular Taxonomy of Breast Cancer International Consortium) is a Canada-UK project which aims to classify breast tumours [29]. Further subcategories in METABRIC help determine the optimal course of treatment and are based on molecular signatures. Patients are assigned to different groups, according to:
• ER expression: two immuno-histo-chemistry sub-types (ER+ and ER-), and
• PAM50 test: five intrinsic gene-expression sub-types (LumA, LumB, HER2-enriched, basal-like, normal-like).

We use both features as target variables in the classification tasks. For solving a binary classification problem, we predict estrogen receptor feature (i.e., ER); and for solving a multi-class classification problem, we predict PAM50 feature.

1.1 Contributions

The core of this thesis is the introduction of a novel rule extraction algorithm of the decompositional nature, which is able to extract rules from deep neural networks. The extracted rules mimic the behaviour of the neural network and therefore explain their decisions. The whole research and the implementation of the algorithm were guided by the following research questions:

• The aim of interpretability is to describe a model in a way that is understandable to humans [16]. Can we make a deep neural network more interpretable?

• A model is called explainable if it is able to summarise the reasons for its behaviour, gain the user's trust, or produce insights about the causes of its decisions [16]. How can we explain the predictions of deep neural networks?

• Can we explain deep neural networks’ decisions by taking into account all their neurons in an efficient way?

Our contribution in this thesis is a rule extraction algorithm, which can extract rules from deep neural networks. Its novelty lies in the fact that it is efficiently able to deal with multi-class classification problems. The algorithm’s performance in terms of how good do extracted rules represent the network’s behaviour is better than that of the pedagogical baseline. Our algorithm outperforms the decompositional baseline by producing shorter and more accurate rules that better mimic the corresponding neural network.

Our algorithm can be useful in applications, where there is a need for understanding the reasons behind the predictions, for example, in healthcare. It can be used to explain decisions of the clinical decision support systems, by making it easier for clinicians to decide if they agree with the system's decision or not, because they have some insight into the reasons that lead to the decision.

1.2 Thesis structure

This thesis is divided into six chapters. In Chapter 2, we give an overview of the related work, describe relevant algorithms and define some basic terms. Chapters 3 and 4 are the core of the thesis. In Chapter 3, we describe our algorithm and provide its pseudocode. We discuss the design and the implementation of the experimental part, and describe a METABRIC data set, which we use for the evaluation. We define measures that are used for the evaluation of the results. In Chapter 4, we evaluate our algorithm
and compare its performance with two baselines, one of the pedagogical and one of the decompositional nature. In Chapter 5, we analyse our algorithm by providing its worst-case theoretical complexity and then give graph representations of the comparison with the two baselines. We conclude with some suggestions for future improvements of the work presented in this thesis, in Chapter 6.
2 Background and literature review

The main question of this thesis is: "How can we make neural networks more interpretable?" In this chapter, we give an overview of works that have been dealing with answering this question.

First, we outline one particular application of neural networks, where the explainability is of paramount importance, that is, clinical decision support systems (CDSS). Next, we discuss the field of deep learning, which is the foundation of CDSS, in Section 2.2. Then, we introduce neural networks, in particular, deep feedforward neural networks. We present the basic idea of how networks work and define some basic terminology. In Section 2.4, we look at rule extraction as an approach to explain neural networks. We outline three approaches to rule extraction from neural networks and present examples for each of them. We describe IF-THEN rules and present measures for evaluating a rule set in Section 2.5. The remaining of Chapter 2 includes descriptions of four algorithms, relevant for our work: CRED, DeepRED, C4.5, and C5.0.

2.1 Clinical decision support systems

A clinical decision support system (CDSS) is an information system with the intention to improve clinical decision making, for example, diagnosis support, treatment, therapy, and prevention of drug toxicity. It combines the characteristics of individual patients with a computerised knowledge base and delivers patient-specific recommendations to the clinician. There are two main types of CDSS: knowledge-based systems and non-knowledge-based systems that use machine learning and other statistical pattern recognition techniques [5].

Knowledge-Based Clinical Decision Support Systems Most systems consist of three parts: knowledge base, the inference or reasoning engine, and a mechanism to communicate with the user. The knowledge base includes compiled information that is often in the form of IF-THEN rules, but sometimes consists of probabilistic associations of signs and symptoms with diagnoses, or known drug-drug interactions. The inference engine contains formulas, which combine the rules or associations in the knowledge base with real patient data. The last part, a communication mechanism, enables us to input the patient data into the system and get the system’s output to the clinician, who will make the actual decision [5].

Non-Knowledge-Based Clinical Decision Support Systems Unlike knowledge-based systems, non-knowledge-based systems do not have the access to the knowledge provided by the human expert but learn from examples. Artificial neural networks represent one type of non-knowledge-based systems [5]. They have become widely used for different biomedical applications [36, 7]. Studies revealed that CDSS based on machine learning and pattern recognition methods could be even more accurate than the average clinician in diagnosing the targeted diseases [30].

The subject of our research are non-knowledge-based CDSS, which use neural networks. But these are black-box models, unable to explain their predictions. Because of that, health care providers are often reluctant to accept and use machine recommendations without the additional explanation [51]. Therefore, many researchers started exploring different approaches to explain the mechanism by which the artificial neural
networks operate.

We discuss deep neural networks and explainability methods on them next.

### 2.2 Deep learning

In artificial intelligence, many tasks that are easy for people turn out to be hard to describe in a formal way, for example, recognising spoken words or faces in images. Researchers have found a solution for this by enabling computers to learn from experience and understand the world in terms of the hierarchy of concepts, where each concept is defined through its relation to simpler concepts. The computer can then learn complicated concepts by constructing them out of simpler ones. If we illustrate these concepts with a graph, they are built on top of each other, constructing a deep graph, with many layers. Therefore, in artificial intelligence, this approach is called deep learning [18].

Deep learning has achieved many notable successes in recent years. Deep learning methods use deep neural networks (DNNs). These are complex neural networks with more than one hidden layer. They are based on biological neural networks and are capable of identifying complex nonlinear relationships between input and output data. Because they are very accurate at making predictions, we use them for a variety of applications, for example, speech recognition [22, 39, 1], natural language processing [9, 45], automated driving [47, 43], and medical research [25]. A computer program based on DNNs, for the first time in history, defeated a human professional player in the complex strategic game Go in 2015 [42].

Despite their high accuracy, DNNs have a major disadvantage. Their lack of transparency has prevented them from being widely used in safety and safety critical systems, for example, healthcare. In other words, they are not able to provide explanations for their decisions, so it is not clear what information in the input data makes them arrive at their decision. Therefore, we usually describe deep neural networks as black boxes.

Next, we describe the principle by which neural networks operate and define terminology that we use later.

### 2.3 Deep feedforward neural networks

Deep feedforward neural networks (also called multilayer perceptrons) are one of the essential deep learning models, and their goal is to approximate some function $f^*$. In particular, if some classifier $y = f^*(x)$ maps $x$ to $y$, then a feedforward neural network defines a mapping $y = f(x; \Theta)$ and learns such values of the parameters $\Theta$ that derive the best function approximation. These networks are called feedforward because information flows only in one direction: through the function being evaluated from $x$, through the intermediate computations that define $f$, and lastly to the output $y$. A network is associated with a directed acyclic graph, which delineates how different functions are composed together. For example, five functions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, $f^{(4)}$, and $f^{(5)}$ define $f(x) = f^{(5)}(f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x)))))$. Then the first layer $f^{(1)}$ is called the input layer of the network, $f^{(2)}$ is called the second layer, and so on. The last layer, in this case $f^{(5)}$, is called the output layer. All layers between the input layer and the output layer
are called hidden layers. The total length of the chain determines the depth of the network. If a network consists of more than one hidden layer, it is called a deep neural network [18]. Fig. 1 shows an example of such a network.

In the process of the neural network’s training, we try to match \( f(x) \) to \( f^*(x) \). Each training example \( x \) has the corresponding label \( y \approx f^* \). The set of training examples directly states that the output layer must produce a value close to \( y \), but it does not specify the behaviour of other layers. Therefore, the learning algorithm must decide how to make use of these layers to best approximate \( f^* \). Due to the indirect influence of data on these layers, they are called hidden layers [18]. Fig. 1 illustrates the general structure of a feedforward neural network, which consists of the input layer with red input neurons on the far left, the output layer consisting of green neurons at the right, and hidden layers with yellow neurons in between.

Russel and Norvig [37] describe an artificial neural network as a set of nodes (units, neurons) connected by directed links. The purpose of a link from node \( i \) to node \( j \) is to transmit the activation \( a_i \) from \( i \) to \( j \). Each link has a corresponding weight \( w_{i,j} \) and each node has a corresponding dummy input \( a_0 \) called bias. Fig. 2 illustrates a simple neuron’s mathematical model for node \( j \). Its activation value \( a_j \) is determined as follows. First, a weighted sum of the inputs is computed:

\[
\text{in}_j = \sum_{i=0}^{n} w_{i,j}a_i
\]  

(2.1)

Then the activation function \( g \) is applied to this sum:

\[
a_j = g(\text{in}_j) = g(\sum_{i=0}^{n} w_{i,j}a_i)
\]  

(2.2)
The role of the activation function $g$ is to introduce non-linearity into the neural networks, which ensures their ability to represent nonlinear functions. Some of the widely used activation functions are:

- **sigmoid**: $f(x) = \frac{1}{1 + e^{-x}}$,
- **rectified linear units (ReLU)**: $f(x) = \max(0, x)$, and
- **softmax**: $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$ for $i = 1, \ldots, C$ and $x = [x_1, \ldots, x_C]$.

Softmax activation function is applied to the vector rather than on a specific value, as is the case with sigmoid and ReLU functions, and is often used on the output layer of the neural network, because it ensures that the values of the output vector sum up to one.

We now know how information flows through the neural network. But how does the neural network learn? For that, it uses a back-propagation algorithm, which uses examples from the training set to improve classifications.

If the output layer of the network contains $k$ output neurons, the network’s output is a vector with $k$ values. We denote it as $h_w$, and a vector with target outputs as $y$. When a training example $x$ is passed to the network, each of the input neurons gets assigned the value of the corresponding input feature. This information then flows through the network, where values of neurons are calculated as defined in Equation (2.2). After $h_w$ is calculated, the algorithm uses some measure to calculate the loss across the $k$ components of the error vector $y - h_w(x)$. The goal is to minimise this error by updating the weights of the neural network. Let us recall that $w_{ij}$ represents a weight from neuron $i$ to neuron $j$. If $Err_j$ is the $j$-th component of the error vector, then we define a modified error as $\Delta_j = Err_j \times g'(in_j)$ and the weight-update rule as

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j \quad (2.3)$$

where $\alpha$ is a fixed learning rate. Essentially, a neural network is trained by updating the weights of the connections between neurons [37]. Fig. 3 provides the algorithm’s pseudo-code in detail.
2.3.1 Definitions

Classification problem Classification is a classic supervised learning task. A typical example is the spam filter, where the model is trained with examples of e-mails together with their class (spam or not spam), and it must learn how to classify new e-mails. Data instances (in this case, e-mails) consist of attributes that describe them (e.g., length of the sender’s e-mail address, length of an e-mail, and whether or not the e-mail has an attachment). If the number of possible classes is two, we have a binary classification problem. In the case of a greater number of classes, we are talking about multi-class classification [15].

Classifier A classifier is a model, constructed to predict class labels, used for solving some specific classification problem [21]. Our thesis, for example, combines two classifiers, that is, deep feedforward neural networks and IF-THEN rules.

k-fold cross validation When working with a small data set, we do not want to split examples into test and validation sets, so cross-validation can be performed. At the beginning, the data is split into k complementary subsets (i.e., each example appears in exactly one subset). We repeat the following process k times: we take each unique subset as a test set and a union of the remaining subsets as a training set, fit the model on the training set and evaluate it on the test set. This way, each example in the original data set is used to train the model k – 1 times, and one time is used for testing [15].
Neoadjuvant therapy Preoperative treatment with the aim of shrinking the tumour before the main treatment [26].

We familiarised ourselves with the area of deep learning and deep feedforward neural networks. Next, we present rule extraction as an approach to explainability of neural networks, which is the focus of this thesis, and give a short overview of the existing approaches.

2.4 Approaches to rule extraction

A trustful model has two important characteristics besides predictive performance: interpretability and explainability. The aim of interpretability is to describe the model in a way that is understandable to humans. Therefore, a model is called interpretable if it can create descriptions that are simple enough for people to understand. In order to completely trust a black-box model, it must also be explainable. All explainable models are interpretable by default, but the reverse does not always hold. A model is called explainable if it is able to summarise the reasons for neural network's behaviour, gain the user's trust, or produce insights about the causes of its decisions [16]. Consequently, artificial neural networks, despite their good performance, can be quite limited in some applications, such as clinical decision support systems, where there is a need for decisions’ explanations. There have been many attempts to interpret or explain neural networks. Fürnkranz et al. [14] argue that rules offer the best trade-off between human and machine understandability. For that reason, in this thesis, we want to explore rule extraction as a way to explain neural networks. This section provides an overview of existing rule extraction algorithms and gives a few examples.

Rule extraction transforms a black-box model into a white-box model by approximating neural network's internal knowledge into a set of symbolic rules [46]. There exist three approaches for rule extraction from neural networks: pedagogical, decompositional, and eclectic [2].

The pedagogical approach considers a neural network as a black box, so it ignores its inner structure and maps input features directly into the output layer [2]. Let us mention a few pedagogical algorithms that deal with rule extraction from different perspectives. Augasta and Kathirvalavakumar have introduced the algorithm RxREN, which uses a reverse engineering technique to prune the insignificant input neurons and to generate the classification rules only with significant input neurons [3]. TREPAN, the algorithm proposed by Craven and Shavlik, produces rules in a form of decision trees using sampling and queries [11]. Saad and Wunsch have presented the algorithm HYPINV, which relies on network inversion. In other words, it calculates the neural network's input, which produces the desired output [38]. In general, these methods are expected to be faster than decompositional algorithms because they ignore the network's inner structure. On the other hand, this also leads to lower fidelity (how well the algorithm can mimic a neural network's behaviour) of the pedagogical algorithms.

Algorithms of the decompositional approach take into account the whole neural network: input and output layers, as well as the network's internal structure, represented by hidden layers. Tsukimoto introduced the polynomial algorithm that can be applied to different kinds of neural networks, such as multilayer perceptron and
recurrent neural network, whose output function is monotone, for example, sigmoid function [48]. Setiono and Leow have presented an algorithm for fast extraction of rules from neural networks, called FERNN. It is applicable on feedforward neural networks with one hidden layer and can generate both M-of-N and IF-THEN rules (see Section 2.5) using relevant hidden units and relevant network connections from input to hidden layer [41]. Using classification algorithm C4.5 (see Section 2.8), Sato and Tsukimoto have presented the algorithm CRED (see Section 2.6), which decomposes a neural network using decision trees and obtains rules by merging the rules extracted from each tree [40]. The modified version of CRED, algorithm DeepRED (see Section 2.7), is the first decompositional algorithm that is able to extract rules from deep neural networks [52]. Existing decompositional algorithms have some disadvantages, namely they are either applicable only on shallow neural networks or are very time and memory expensive.

Algorithms of the eclectic approach have features of both, pedagogical and decompositional approaches, which have some benefits and disadvantages. Regarding computational complexity, pedagogical algorithms are simpler, because they take into consideration only input and output layers of the neural network. Consequently, they lack a big part of the information from the network’s inner structure. Decompositional algorithms, on the other hand, split the network into neuron levels and aggregate the results obtained from each neuron to represent the neural network as a whole. This requires large time and memory resources, which increases significantly with greater number of hidden layers. If we could better manage time and memory complexity, decompositional algorithms better explain neural networks, because they take their whole structure into consideration.

In this thesis, we present the rule extraction algorithm of the decompositional nature, with which we fill the gap of scalability to deep neural networks that can deal with multi-class classification. We introduce our algorithm in Section 3.2, but first, we introduce IF-THEN rules and describe how we can use them for classification.

### 2.5 IF-THEN rules

There are many different notations to represent a rule. For the purpose of this thesis, we are interested in the rules of the form:

\[
\text{IF conditions THEN class } = X.
\]

Conditions are conjunction of logical tests on the attributes, for example,

\[
test_1 \text{ AND } test_2 \text{ AND } \ldots \text{ AND } test_L.
\]

For the rule to be applicable (i.e., considered), all conditions must be satisfied. The rule length is defined as the number of tests in the condition part of the rule (in this case, \(L\)). Tests (or terms) are of the form \(A_i = v\) for discrete attributes, and \(A_i \leq v\) or \(A_i > v\) for continuous attributes. The conclusion of the rule follows the word "THEN" and contains a class value. Basically, the rule predicts class \(X\) for all examples that satisfy all \(L\) conditions [14]. In other words, an example is covered by a rule \(r\) if all conditions are satisfied for that concrete example.
As a simple example, suppose there are five attributes $A_1, A_2, A_3, A_4, A_5$, two examples with their attributes stored in a vector: $e_1 = [2, 7, 11, 15, 8]$ and $e_2 = [6, 36, 9, 23, 4]$, and a rule $r$: IF $A_1 \leq 12$ AND $A_4 > 21$ AND $A_3 \leq 17$ THEN class $= c_3$. Then the rule $r$ covers (is applicable to) the example $e_2$, but it does not cover the example $e_1$. For the example $e_2$, the class $c_3$ is predicted.

There exist several measures to evaluate a rule $r$. Given a class-labelled data set of examples, let $|D|$ be the number of examples in $D$, $n_{covers}$ be the number of examples covered by $r$, and $n_{correct}$ be the number of examples correctly classified by $r$. Then we define the coverage and accuracy of a rule $r$ as:

$$coverage(r) = \frac{n_{covers}}{|D|}$$ (2.4)

$$accuracy(r) = \frac{n_{correct}}{n_{covers}}$$ (2.5)

The percentage of examples that are covered by the rule is called coverage, and the percentage of correctly classified examples out of those covered by the rule is called accuracy [21].

For solving a classification problem in practice, we never use only one rule, but multiple IF-THEN rules combined into a rule set $R$. The classification for a previously unseen example depends on whether the rules are in some order or not. If the rules are ordered, the first rule that covers the example is chosen to classify it; if the rules are not ordered, all rules are evaluated and predictions of those that cover the example are used. However, it can happen that they make conflicting predictions. The classical solution is a voting mechanism to acquire the final prediction. Whether the rules are ordered or not, it can happen that no rule covers the new example. In this case, a default rule is used, which usually predicts the majority class of uncovered training examples [14]. The default rule has an empty condition and is evaluated only if no other rule covers the example [21].

To define measures for evaluating rule sets as a whole, we need to introduce a few notations, presented in Table 1. Given a specific target class, examples are either positive or negative, and they are covered (predicted positive) or not covered (predicted negative) by a set of rules $R$. Correctly predicted positive examples are called true positives, correctly predicted negative examples are called true negatives, positives incorrectly predicted as negatives are called false negatives, and negatives incorrectly predicted as positives are called false positives [14]. The number of examples that fall into these sets are denoted as $TP$, $TN$, $FN$, and $FP$, respectively. This notation is summarised in Table 1. Usually, we denote all positive examples as $P$, and all negative examples as $N$. In other words: $P = TP + FN$ and $N = FP + TN$.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Covered</th>
<th>Not Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Negative</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Table 1: Notation for sets of covered and uncovered positive and negative examples.

The classification accuracy of the rule set is defined as "the percentage of the total number of correctly classified examples in all classes relative to the total number of
tested examples" [14]. This should not be confused with the accuracy for one rule, defined in Equation (2.5). For a binary classification problem, the classification accuracy is calculated as:

\[
\text{accuracy}(R) = \frac{TP + TN}{P + N}
\]  

(2.6)

As an alternative, the results are often given with the classification error, which is defined as:

\[
\text{error}(R) = 1 - \text{accuracy}(R) = \frac{FP + FN}{P + N}
\]  

(2.7)

Comparing IF-THEN rules to neural networks, it is fair to say that rules are far more straightforward. With explicitly given conditions, we know exactly how the predicted class is dependent on the input features, whereas the neural network only gives us the prediction. We can look at the activation values for each example of neurons on all layers, but this enormous amount of information does not directly provide an explanation.

In the remainder of Chapter 2, we discuss a decompositional algorithm CRED, which represents the base for DeepRED that inspired the implementation of our rule extraction algorithm.

### 2.6 CRED

Sato and Tsukimoto have presented a decompositional algorithm CRED (Continuous/discrete Rule Extractor via Decision tree induction) [40], which extracts rules from neural networks with one hidden layer, that contain both continuous and discrete literals\(^1\). First, CRED decomposes a neural network using decision trees based on the activation patterns of hidden and output units, and input and hidden units. Then it generates final rules by merging the rules extracted from each tree. CRED consists of five steps, shown in Fig. 4.

1. Set a target class and set the target pattern of output units.
2. Build a hidden-output tree, extract intermediate rules, and decompose the network into constituent functions.
3. Build an input-hidden tree and extract input rules for each of the functions.
4. Acquire total rules by substituting the input rules for the intermediate rules.
5. Merge the total rules.

In step 1, the algorithm decides what will be the question (query) to a neural network. The type of question depends on whether the neural network is trained for a continuous or a discrete class variable. Continuous domains are either divided into user-defined intervals or are discretised using clustering, which enables to automatically generate ranges by using cluster boundaries.

Step 2 uses the activation pattern of hidden units as attribute data and the discretised pattern produced in the previous step as class data to build a hidden-output

\(^1\)In this section only, we refer to attributes as literals, as the authors of CRED did.
decision tree. Because each leaf corresponds to one rule, exclusive intermediate rules are extracted from the decision tree. Each rule is then simplified by removing useless literals and eliminating rules that are included by another rule. Here, we observe the boundaries used in the literals of the intermediate rules. In Fig. 4, h1 represents the first unit (neuron) and h2 the second unit in the hidden layer. The corresponding boundaries are 0.6 and 0.5 for h1 and h2, respectively. Therefore, some function of h1 will distinguish the activation pattern of the previous layer by 0.6. This step also creates a new query: "what is the condition that makes the activation values of h2 greater than 0.5?" and discretises the activation values of h2. The same process is repeated for all boundaries.

Figure 4: The CRED algorithm (adapted from [40]).
Step 3 builds input-hidden decision trees where each of them corresponds to a query generated in the previous step. Here, the input variables act as the attribute variables and the discretised pattern created in step 2 is the class variable. Similar to the previous step, input rules are extracted from each input-hidden tree. They are then simplified, and redundant rules are eliminated. A rule is called redundant if it is included in another rule.

In step 4, the input rules are substituted for intermediate rules, and the result are total rules, which describe the relationship between the input pattern and the target query. Each total rule gets simplified, and possible redundant rules are eliminated.

Step 5 merges total rules to get simplified rules at the end. For this, the algorithm uses some merging measure, for example, J-measure. For rule \( X \rightarrow Y \), let \( p(X), p(Y), p(XY) \) denote the fractions of data satisfying condition \( X, Y, XY \), respectively, and \( p(Y|X) = \frac{p(XY)}{p(X)} \). J-measure is then defined as:

\[
J(X, Y) = p(X)p(Y|X)\log_2 \frac{p(Y|X)}{p(Y)} + p(X)(1 - p(Y|X))\log_2 \frac{1 - p(Y|X)}{1 - p(Y)} \tag{2.8}
\]

Authors also propose a new simplification method, but that is out of the scope of this thesis.

The advantage of the CRED algorithm is that it is applicable to both discrete and continuous variables. On the other hand, it can only be used for extracting rules from neural networks with one hidden layer. Zilke et al. [52] have proposed an algorithm DeepRED, a modified version of CRED, that can also be used on deep neural networks, which we describe next.

2.7 DeepRED

According to [52], DeepRED (Deep neural network Rule Extraction via Decision tree induction) is the first algorithm that has been explicitly tested on the task of extracting rules from deep neural networks. It is based on the algorithm CRED and it uses a decompositional approach to extract rules. DeepRED's pseudo-code is shown in the Fig. 5.

Without loss of generality, we consider dealing with a classification problem with \( n \) classes and extract rules from a deep neural network with \( k \) hidden layers \( h_1, \ldots, h_k \). DeepRED repeats the process of extracting rules \( n \) times, once for each of the classes. First, it uses a classification algorithm C4.5 [34] to build decision trees that have split points on the activation values of neurons in the last hidden layer and the corresponding classifications in their leaves. The result of this step is a rule set \( R_{h_k \rightarrow o} \), which contains rules describing the output layer \( o \) in terms of the neurons in the last hidden layer \( h_k \).

In the next step, CRED would refer to the input layer, but DeepRED processes the next shallower layer \( h_{k-1} \). For every term (e.g., \( h_{k,1} > 0.5 \)) that appears in one of the rules in \( R_{h_k \rightarrow o} \), it applies C4.5 to generate decision trees describing layer \( h_k \) in terms of the neurons in the second to last hidden layer \( h_{k-1} \). This gives us a rule set \( R_{h_{k-1} \rightarrow h_k} \).

DeepRED proceeds in the same way until it generates a rule set \( R_{i \rightarrow h_1} \). This rule set contains rules that describe terms in the first hidden layer \( h_1 \) by terms consisting of the input features in layer \( i \). In each step, the algorithm checks if a decision tree to
describe a term has already been extracted. In that case, it just copies a resulting tree. With this check, the algorithm prevents performing redundant C4.5 calls to build trees that were already generated.

So far, the algorithm has produced rule sets $R_{i\rightarrow h_1}, R_{h_1\rightarrow h_2}, \ldots, R_{h_{k-1}\rightarrow h_k}, R_{h_k\rightarrow o}$. In other words, we have rules that describe each layer by their respective preceding layer. In the end, we want to have a rule set $R_{i\rightarrow o}$, describing the output layer in terms of the input features. Therefore, as the next step, the algorithm performs merging. It does so in the layer-wise manner in reverse order (from the output layer towards the input layer).

First, the algorithm merges rule sets $R_{h_k\rightarrow h_{k-1}}$ and $R_{h_{k-1}\rightarrow h_k}$ by substituting the terms in $R_{h_k\rightarrow o}$ by regarding rules in $R_{h_k\rightarrow h_{k-1}}$. The result is a rule set $R_{h_{k-1}\rightarrow o}$. If it contains any unsatisfiable intermediate rules or redundant terms, they are removed. Next, $R_{h_{k-2}\rightarrow h_{k-1}}$ and $R_{h_{k-1}\rightarrow o}$ are merged, resulting in a rule set $R_{h_{k-2}\rightarrow o}$. The algorithm proceeds in the same manner through all layers until it generates a rule set $R_{i\rightarrow o}$.

Let us illustrate the merging of two rule sets on a simple example. A rule set $R_{h_i\rightarrow o}$

---

Figure 5: Pseudo-code of the DeepRED algorithm [53].
is created by merging rule sets $R_{h_i \rightarrow h_{i+1}}$ and $R_{h_{i+1} \rightarrow o}$. If there are two rules in $R_{h_{i+1} \rightarrow o}$:

A: IF $h_{i+1,3} \leq 0.7$ THEN $X_2$
B: IF $h_{i+1,9} \leq 0.2$ AND $h_{i+1,4} > 0.8$ THEN $X_1$

then for each of the terms $h_{i+1,3} \leq 0.7$, $h_{i+1,9} \leq 0.2$, and $h_{i+1,4} > 0.8$ (i.e., terms in $R_{h_{i+1} \rightarrow o}$), rules that express them in terms of neurons in the layer $h_i$ were already built in the previous steps of the algorithm. The union of these rules creates a rule set $R_{h_i \rightarrow h_{i+1}}$. What follows are rules for each of the terms that appear in $R_{h_{i+1} \rightarrow o}$. Their prediction tells us whether the term applies or not.

Rules for $h_{i+1,3} \leq 0.7$:

C: IF $h_{i,6} > 0.1$ AND $h_{i,9} \leq 0.4$ THEN $h_{i+1,3} \leq 0.7$
D: IF $h_{i,1} \leq 0.8$ AND $h_{i,6} \leq 0.1$ THEN $h_{i+1,3} \leq 0.7$
E: IF $h_{i,8} > 0.3$ AND $h_{i,9} \leq 0.4$ AND $h_{i,7} \leq 0.1$ THEN $h_{i+1,3} > 0.7$

Rules for $h_{i+1,9} \leq 0.2$:

F: IF $h_{i,3} \leq 0.5$ THEN $h_{i+1,9} \leq 0.2$
G: IF $h_{i,5} \leq 0.7$ AND $h_{i,3} > 0.5$ THEN $h_{i+1,9} \leq 0.2$
H: IF $h_{i,4} \leq 0.8$ THEN $h_{i+1,9} \leq 0.2$
I: IF $h_{i,1} > 0.6$ AND $h_{i,6} \leq 0.2$ THEN $h_{i+1,9} > 0.2$

Rules for $h_{i+1,4} > 0.8$:

J: IF $h_{i,3} \leq 0.2$ AND $h_{i,6} \leq 0.5$ THEN $h_{i+1,4} \leq 0.8$
K: IF $h_{i,8} > 0.4$ AND $h_{i,7} > 0.6$ THEN $h_{i+1,4} > 0.8$
L: IF $h_{i,2} \leq 0.1$ THEN $h_{i+1,4} > 0.8$

The merging is done by replacing terms in $R_{h_{i+1} \rightarrow o}$ (i.e., $h_{i+1,3} \leq 0.7$, $h_{i+1,9} \leq 0.2$, and $h_{i+1,4} > 0.8$) with their corresponding rules in $R_{h_{i} \rightarrow o}$, by taking into account all possible combinations. First, term $h_{i+1,3} \leq 0.7$ in rule A is replaced by rules C and D (see predictions of the rules C, D, and E), which creates rules:

M: IF $h_{i,6} > 0.1$ AND $h_{i,9} \leq 0.4$ THEN $X_2$
N: IF $h_{i,1} \leq 0.8$ AND $h_{i,6} \leq 0.1$ THEN $X_2$

Next, all possible combinations of corresponding rules for $h_{i+1,9} \leq 0.2$ and $h_{i+1,4} > 0.8$ are made, creating rules:

O: IF $h_{i,3} \leq 0.5$ AND $h_{i,8} \leq 0.4$ AND $h_{i,7} > 0.6$ THEN $X_1$
P: IF $h_{i,5} \leq 0.7$ AND $h_{i,3} > 0.5$ AND $h_{i,8} \leq 0.4$ AND $h_{i,7} > 0.6$ THEN $X_1$
Q: IF $h_{i,4} \leq 0.8$ AND $h_{i,8} \leq 0.4$ AND $h_{i,7} > 0.6$ THEN $X_1$
R: IF $h_{i,3} \leq 0.5$ AND $h_{i,2} \leq 0.1$ THEN $X_1$
S: IF $h_{i,5} \leq 0.7$ AND $h_{i,3} > 0.5$ AND $h_{i,2} \leq 0.1$ THEN $X_1$
T: IF $h_{i,4} \leq 0.8$ AND $h_{i,2} \leq 0.1$ THEN $X_1
Rules M-T make up a rule set $R_{h_i \rightarrow o}$ with the conditions on neurons in the $i$-th hidden layer, mapping to the output layer (classes $X_1$ and $X_2$).

The rule set $R_{i \rightarrow o}$ contains rules with conditions on the input features, which predict classes in the output layer. We emphasise once again that this algorithm generates rules for each class separately, one after the other, rather than generating them all in one pass. Its complexity is therefore linearly dependent on the size of the output layer.

We discussed how the DeepRED algorithm extracts rules from neural networks. Next, we take a closer look into the classification algorithm C4.5, which DeepRED uses for generating decision trees that connect two consecutive layers.

### 2.8 C4.5

Classification models take as an input a collection of cases, where each case belongs to one class and is determined by its values for a fixed set of attributes. The output is a classifier that can predict which class some particular case belongs to [28]. One of the classification models are decision trees.

An algorithm C4.5 is used to generate decision trees. It was developed by Quinlan in 1993 [34] and it is an extension of Quinlan's previous algorithm ID3 [33]. C4.5 builds a decision tree from a set of training data using the information entropy. If we have a discrete random variable $X$ with possible values $x_1, x_2, \ldots, x_n$, and $P(x_i)$ is the probability of $x_i$, then the information entropy is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2(P(x_i))$$  \hspace{1cm} (2.9)

Given a set of cases $S$, the algorithm C4.5 applies the divide-and-conquer algorithm to build an initial tree. If all cases in $S$ belong to the same class or $S$ is small, then the tree is declared a leaf. The leaf is labelled with the class, which occurs most frequently in $S$. If the algorithm does not generate a leaf, it chooses a test on a single attribute. A test may have multiple outcomes and represents the root of the tree with one branch for each possible outcome of the test. A set $S$ is then partitioned into subsets, according to each case. The same procedure is then applied on each of the subsets [28]. Let us illustrate this process on a simple example. Given a set of cases, presented in Table 2, an algorithm C4.5 builds a decision tree, illustrated in Fig. 6.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 2: An example set with seven cases, three attributes (i.e., X, Y, and Z) and two classes (i.e., A and B).
First, C4.5 splits the data on the attribute Y with two outcomes (i.e., 0 and 1), because Y makes the clearest distinction between cases of different classes. After splitting the data, C4.5 makes a leaf in the case when Y equals 0, because all cases in that subset belong to the same class, that is, B. Regarding the other outcome, that is, when Y equals 1, C4.5 splits the corresponding subset on the attribute Z. Both outcomes provide subsets, which contain cases that belong to the same class, so C4.5 generates two leaves and returns the tree in Fig. 6. For clarity, the nodes are represented as the classes of cases in corresponding subsets. C4.5 labels two leaves with class B and one leaf with class A.

How does C4.5 decide which attribute to split on? It chooses the attribute of the data that splits the set of training cases in the best way. This means that the generated subsets are heading towards single-class cases. For this, C4.5 uses a criterion, called normalised information gain (see Equation (2.13)), which is defined as the difference in entropy. If \( S \) is a set of \( |S| \) cases, where each case belongs to one of the classes \( C_1, C_2, \ldots, C_k \), and \( P(C_i) \) is the probability of \( C_i \), then the average information needed to identify the class of a case in \( S \) is calculated with Equation (2.9) as:

\[
H(S) = -\sum_{i=1}^{k} P(C_i) \log_2(P(C_i)) \tag{2.10}
\]

Now, let us consider splitting a set \( S \) in accordance with the \( j \) outcomes of the attribute \( X \). The expected information requirement is calculated as the weighted sum over the subsets \( S_1, S_2, \ldots, S_j \):

\[
H_X(S) = \sum_{i=1}^{j} \frac{|S_i|}{|S|} \times H(S_i) \tag{2.11}
\]

The information gain of attribute \( X \) is defined as:

\[
gain(X) = H(S) - H_X(S) \tag{2.12}
\]

and it measures the information that is gained by partitioning a set \( S \) with respect to the attribute \( X \). By simply selecting the attribute that has the maximum information gain, we favour attributes with more outcomes. In order to avoid this, C4.5 uses a normalised information gain, which is calculated as:

\[
normalised\ gain(X) = \frac{gain(X)}{split\ info(X)} \tag{2.13}
\]
where \( \text{split info} \) represents the normalisation and is calculated as:

\[
\text{split info}(X) = -\sum_{i=1}^{I} \frac{|S_i|}{|S|} \times \log_2 \frac{|S_i|}{|S|}
\]

For the particular node, C4.5 chooses the attribute with the highest normalised information gain [34].

After the initial tree is built, C4.5 performs the pruning in order to avoid overfitting. The pruning algorithm proceeds from the leaves to the root and is based on the estimation of an error rate. When considering a subtree, C4.5 compares the sum of estimated errors of the branches with the estimated error if the subtree gets replaced by a leaf. If the former error is greater than the latter error, a corresponding subtree is pruned [28].

Beside the final decision tree, C4.5 can also generate a list of rules of the form: IF \( \text{test}_1 \) AND \( \text{test}_2 \) AND \ldots THEN class C. The rule set is based on the unpruned decision tree and is generated by grouping together the rules for each class. First, prototype rules are defined, simply by following each path from the root of the tree to the leaves. The conditions of one particular rule are the outcomes along the path and the class of the rule represents the label of the leaf. However, if the path to each leaf is directly transformed into a rule, the resulting set of rules classifies cases exactly as the decision tree does. Furthermore, since this produces one rule for every leaf, a set of rules is no simpler than the tree. But, by removing conditions one by one, the rules get simplified [28]. After pruning rules one by one, the set of rules associated with each class must be processed separately to reduce and order the rules. Redundant rules get removed using the MDL principle, which says that if there are two rule sets with the equivalent performance, the simpler one is chosen. Rules within each class are ordered from most to least accurate. Then the classes are ordered based on the accuracy and a default class is chosen. When classifying a new case, each rule is evaluated until one meets all necessary conditions. The predicted class corresponds to the class of that first rule. If no rule applies, then the case is assigned to a default class [24].

The number of rules is much smaller than that of the leaves from the pruned decision tree, but the process of generating the rules is very time and memory demanding. The following section introduces C4.5’s successor, a commercial system named C5.0, which generates more accurate and less time and memory demanding rules [35].

### 2.9 C5.0

As the improved version of the C4.5 classification algorithm, C5.0 has some additional features, for example, boosting and different costs for different types of errors, but the algorithm’s core is very similar to that of C4.5. The improved version is also able to produce decision trees as well as a set of rules. According to Kuhn and Johnson, the existing literature on C5.0 is mostly coming from the evaluation of the program’s source code, so we discuss the algorithm based on their book [24].

Due to a few improvements, decision trees generated by C5.0 are likely to be smaller than trees generated by C4.5. The classification algorithm C5.0 creates rules with the similar process than its predecessor did: it generates the initial tree, transforms it into rules (each path from root to the leaf represents one rule), which are simplified via
pruning and a global procedure is used on the whole rule set. The technique for pruning conditions within a rule and simplifying the entire rule set is the same as in C4.5, but C5.0 does not order the rules in the end. As an alternative, when C5.0 predicts a class for a new case, it uses all applicable rules. The votes for each class are multiplied by the confidence values, and the class with the greatest value is predicted.

We will take a closer look at one of the C5.0’s features, which we used in our algorithm (see Section 3.2). More features that improve C4.5, for example, boosting and predictor importance, can be found in [24]. C5.0 has an option to remove predictors via the feature called *winnow*. The algorithm discovers the attributes, which have a relationship with the outcome, and only these important predictors are used for creating the final model. To find a subset of important predictors, C5.0 first randomly splits the training set in half and creates the "winnowing tree" in order to evaluate the utility of the predictors. There are two procedures that determine the importance of each predictor.

- If the predictor is not in any split in the winnowing tree, it is viewed as unimportant.
- The other half of the training data, which was not used for creating the winnowing tree, is used to estimate the tree’s error rate. Also, the error rate is estimated without each predictor and compared to the error rate when all predictors are used. In the case that the error rate is improved without the predictor, the corresponding attribute is irrelevant and therefore removed.

After identifying unimportant attributes, C5.0 recreates the tree. If the error rate is worse, the algorithm ignores the winnowing. With this process, C5.0 establishes important predictors. It then proceeds to the training process with the entire training set, but only uses predictors that were not removed during the winnowing procedure.

Revathy and Lawrance compared C4.5 and C5.0 algorithms with an experimental approach [35] and came to the conclusion that C5.0 is more efficient. The rules generated by C5.0 were more accurate and also less time and memory demanding compared to those generated by C4.5.

### 2.10 Summary

Chapter 2 has provided an overview of the motivation for extracting rules from neural networks and how this was done in related research. Section 2.1 presented a particular application of neural networks, that is, clinical decision support systems (CDSS), where the explainability is of paramount importance. Non-knowledge-based CDSS use neural networks, which were described along with deep learning in Sections 2.2 and 2.3. If we were able to explain the neural networks’ decisions, this would lead to the greater use of these black-box models in safety critical applications. Despite their high accuracy, deep neural networks are not transparent and they lack interpretability and explainability. One way of tackling this problem is extracting rules from them. The majority of existing algorithms are only able to extract rules from shallow neural networks, and they either do not take the whole network’s architecture into account or

---

2Kuhn and Johnson [24] refer to attributes as predictors.
are very time and memory expensive. Different approaches, along with their representative algorithms, were described in Section 2.4. We later describe our rule extraction algorithm (see Section 3.2), which uses simple IF-THEN rules, discussed in Section 2.5. If we compare rules to neural networks, it is fair to say that rules are far more straightforward. Sections 2.6 and 2.7 described two decompositional algorithms that can be used to extract rules from neural networks. The first one, CRED, is used to extract rules from neural networks with one hidden layer. The second algorithm, DeepRED, is an extension of CRED and can be used on deep neural networks. Its disadvantages are scalability and, even though it can be used for multi-class classification, it was only tested on binary classification problems. Section 2.8 described a classification algorithm C4.5, used by both CRED and DeepRED to generate decision trees. Our rule extraction algorithm (see Section 3.2) also builds decision trees, but with another algorithm, that is, C5.0, discussed in Section 2.9. With this, we conclude the section of background and literature review and proceed to our rule extraction algorithm.
3 Design and implementation

In Chapter 2, we have shown that there exist multiple approaches that tackle the problem of explaining neural networks. Because of the simplicity of IF-THEN rules, we have focused on rule extraction as a way to interpret neural networks. There are many algorithms implemented for this task, however, to the best of our knowledge, there is only one algorithm that has been used for rule extraction from deep neural networks (see Section 2.7). Inspired by this algorithm, DeepRED, we have implemented a rule extraction algorithm of the decompositional nature, which can extract rules from deep neural networks. Its novelty is being able to deal with multi-class classification problems. Furthermore, it extracts shorter and more accurate rules with a higher level of fidelity than the state-of-the-art decompositional algorithm DeepRED.

This chapter provides information about the design of the experiments and their implementation. Section 3.1 introduces the design and outlines the basic steps of the experiments. Section 3.2 consists of a detailed description of our rule extraction algorithm and its pseudo-code. For better understanding, the algorithm is described together with a running example. The algorithm extracts rules from the activation values of a neural network. Therefore, the first step is to provide trained neural networks on which the algorithm will run. For this, we chose breast cancer data from the METABRIC data set, which is described in Section 3.3. In the following section, we discuss our experimental setup, which consists of three experiments. Extracted rules approximate deep neural network, from which the rules are extracted. Section 3.5 presents measures used for the evaluation of rules, extracted with our algorithm.

First, we give an overview of the experimental part and describe the crucial steps in the following section.

3.1 Design

We have implemented a decompositional rule extraction algorithm, which extracts rules from deep neural networks. It does so by looking at the activation values of a trained neural network. The algorithm proceeds in a layer-wise manner from the network’s output layer towards the input layer. At each step, it generates rules with the classification algorithm C5.0 (described in Section 2.9). These rules represent each layer of the neural network in terms of the neurons in the preceding layer. Afterwards, the algorithm again proceeds layer-wise, starting at the output layer, and merges generated rules. In the end, it returns a rule set, which describes the output layer based on the attributes of the input layer. The detailed description of the algorithm is given in Section 3.2.

To experiment with our algorithm and evaluate the extracted rules, we need a deep neural network and some concrete data to train the network on. In Section 2.1, we described an example of safety critical systems, where the explanation of neural networks is of paramount importance. This has lead us to use a healthcare domain to evaluate our algorithm. In particular, we chose a breast cancer data set METABRIC, described in Section 3.3.

The pipeline for the experimental part is shown in Fig. 7. Let us go through the main steps.
After studying METABRIC data set, we choose to perform two classification tasks. First, we consider a binary classification problem. To show that our algorithm is able to deal with multi-class classification, we then classify for five classes. Overall, we run three experiments:

- binary classification with 1,000 input features,
- binary classification with 2,000 input features, and
- multi-class classification with 2,000 input features and five outputs.

We describe the chosen tasks and features in detail in Section 3.4. There we also define three models for solving each of the listed experiments: $DNN_1$, $DNN_2$, and $DNN_3$, respectively.

Next, we choose the architectures of neural networks that we will use to solve chosen tasks. Experiments, defined in (A), partially determine the structures of neural networks we need to implement to solve these classification problems. In particular, the number of neurons in the input layer equals the number of input features. Similarly, the output layer consists of two neurons for the first two tasks and five neurons for the third task. We decided to experiment with neural
networks with two hidden layers. The general architecture of the networks for experimenting is illustrated in Fig. 8.

Figure 8: The architecture of networks for the three experiments. $DNN_1$ has 1,000 neurons in the input layer and two neurons in the output layer; $DNN_2$ has 2,000 input neurons and two output neurons; $DNN_3$ has 2,000 inputs and five output neurons.

We now perform a grid-search to determine the best number of neurons for hidden layers. For training a neural network, there are other parameters that can be optimised, for example, the number of epochs and batch size. One epoch refers to one pass of all training examples to the neural network, while batch size represents the number of training examples in one pass. The implemented grid-search and its results are described in Section 3.4.

We defined the architectures of neural networks for all three tasks, defined in (A), and chose the data. Now, we train networks on the selected data and make predictions on the test set.

We save the activation values of trained networks along with their predictions.

We use activation values of trained neural networks as an input to our rule extraction algorithm, which returns a set of IF-THEN rules. These rules describe the output layer of neural networks in terms of the input features.
The extracted rules mimic the behaviour of the neural network, whose activation values they were built on. Lastly, we evaluate these rules with measures, defined in Section 3.5.

In the next section, we describe our rule extraction algorithm in detail.

### 3.2 Rule extraction algorithm

Our algorithm extracts rules from deep neural networks, and these rules map the input features of the network to its output. The algorithm uses a decompositional approach, which means that beside input and output layers, it considers the network’s hidden layers as well. The base idea is to use the neural network’s activation values of all neurons on all layers for generating the rules. Therefore, the first step is to train the network.

Let us assume we are solving a multi-class classification problem with \( c \) classes \( \{\lambda_1, \lambda_2, \ldots, \lambda_c\} \), and \( m \) training examples \( x_1, x_2, \ldots, x_m \), where each \( x_j \) has a corresponding class \( y_j \in \{\lambda_1, \lambda_2, \ldots, \lambda_c\} \). The output layer is represented as \( o = o_1, \ldots, o_c \) with as many neurons as there are possible classes. With \( h \) attributes, we represent the input layer as \( i = i_1, \ldots, i_h \). Hidden layers are represented with \( h_i \in \{h_1, h_2, \ldots, h_k\} \). For the input and output layers holds that \( i = h_0 \) and \( o = h_{k+1} \). For input instance \( x_j \) we denote values in the \( i \)-th hidden layer as \( h_i(x_j) \).

Let us recall some terminology from Section 2.5:

- A rule with \( j \) conditions looks like: IF \( t_1 \) AND \( t_2 \) AND \( \ldots \) AND \( t_j \) THEN class = \( X \).

- Rule’s conditions are called terms, and term \( t \) looks like one of the forms: 
  
  \[
  \text{neuron} \leq \text{threshold} \text{ or } \text{neuron} > \text{threshold}.
  \]

The general concept of the algorithm is to handle every hidden layer in reverse order as in CRED and DeepRED (presented in Sections 2.6 and 2.7, respectively). At each step, the algorithm uses classification algorithm C5.0 to extract intermediate rules, which describe the behaviour of the current layer in terms of the neurons in the preceding layer. These rules are extracted from decision trees that are built on the activation values of the corresponding layers (see Section 3.2.1). After extracting rules for each of the two consecutive layers, the algorithm merges the rules and returns a rule set, which describes the output layer based on the attributes of the input layer (see Section 3.2.2). The algorithm’s pseudo-code is given in Algorithm 1.

Our theoretical analysis of the algorithm’s worst-case complexity (discussed separately for building and merging rule sets in Section 5.2) suggests that the algorithm is exponential. In practice, the classification is done only on preselected (due to the C5.0’s winnowing) important features. Also, for reducing computation time and memory, the algorithm removes redundant terms and unsatisfiable rules after every merge (Algorithm 1, lines 13 and 14).

**Example:** We illustrate the functionality of the algorithm on a specific running example of a binary classification problem: predicting ER expression. In this toy example, a neural network’s structure is 100-32-8-2, and we can either predict ER- or ER+, as they are the only values that can appear in the leaves of the final decision tree. Let us denote neurons in hidden layers as \( h_{10}, h_{11}, \ldots, h_{131} \) and \( h_{20}, h_{21}, \ldots, h_{27} \) for the first and
If the activation value of the neuron is less than or equal to 0.3, we give the activation values for layer e.g., the value is greater than 0.3 and 1 (indicating the condition in the term is satisfied, e.g., the value is less than or equal to 0.3). We apply this term on activation values in layer e.g., the value is greater than 0.3 and 1 (indicating the condition in the term is satisfied, e.g., the value is less than or equal to 0.3). We give the activation values for layer \( h_{i-1} \) and calculated target attribute on the activation values for layer \( h_i \) to C5.0 as an input, and C5.0 returns a decision tree.

C5.0 has an optional ability to convert a decision tree into a rule set that consists of an unordered collection of simple IF-THEN rules. As described in Section 2.9, rules

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} Trained neural network \( h_0, h_1, \ldots, h_{k+1} \); training examples \( x_1, x_2, \ldots, x_m \)
\State \textbf{Output:} Set of rules, mapping the input layer to the output layer
\State \( x_1', \ldots, x_m' \leftarrow h_k(x_1), \ldots, h_k(x_m) \)
\State \( y_1', \ldots, y_m' \leftarrow \text{classWhenMax}(h_{k+1}(x_1), \ldots, \text{classWhenMax}(h_{k+1}(x_m)) \)
\State \( R_{h_k \rightarrow o} \leftarrow \text{C5.0} ((x_1', y_1'), \ldots, (x_m', y_m')) \)
\For{each hidden layer \( l = k, k-1, \ldots, 1 \)}
\State \( R_{h_{l-1} \rightarrow h_l} \leftarrow \emptyset \)
\State \( T \leftarrow \text{extractUniqueTermsFromRules} (R_{h_{l-1} \rightarrow h_{l+1}}) \)
\State \( x_1', \ldots, x_m' \leftarrow h_{l-1}(x_1), \ldots, h_{l-1}(x_m) \)
\State \( y_1', \ldots, y_m' \leftarrow h_l(x_1), \ldots, h_l(x_m) \)
\For{each term \( t \in T \)}
\State \( R_{h_{l-1} \rightarrow h_l} \leftarrow R_{h_{l-1} \rightarrow h_l} \cup \text{C5.0} ((x_1', t(y_1')), \ldots, (x_m', t(y_m')) \)
\EndFor
\EndFor
\State \( R_{h_{l-1} \rightarrow o} \leftarrow \text{mergeRules} (R_{h_{l-1} \rightarrow h_l}, R_{h_{l-1} \rightarrow o}) \)
\State \( R_{h_{l-1} \rightarrow o} \leftarrow \text{removeRedundantTerms} (R_{h_{l-1} \rightarrow o}) \)
\State \( R_{h_{l-1} \rightarrow o} \leftarrow \text{removeUnsatisfiableRules} (R_{h_{l-1} \rightarrow o}) \)
\State \( \text{return } R_{l \rightarrow o} \)
\end{algorithmic}
\end{algorithm}

second hidden layer, respectively (the first number represents a layer and the remaining digits tell us which neuron in that layer). The input layer consists of one hundred neurons, denoted as \( i_0, i_1, \ldots, i_{99} \) and the output layer has two neurons. In our examples, the left branch represents the state of conditions being evaluated as true (true branch) and the right one represents the state when the conditions are evaluated as false (false branch). Just for the sake of better understanding of merging rule sets, we present decision trees and rule sets interchangeably in this toy example.

Let us now go through the algorithm in detail.

### 3.2.1 Building the rules

One of the main parts of the algorithm is building decision trees on the activation values of two consecutive layers and transform them into rules. In particular, we want to express layer \( h_i \) in terms of the neurons on the preceding layer \( h_{i-1} \). For this, we use a classification algorithm C5.0 (see Section 2.9). As we describe later, during the algorithm, there will appear terms with conditions on neurons of layer \( h_i \) (e.g., \( h_{i,n} \leq 0.3 \)). We apply this term on activation values in layer \( h_i \) for all training examples (e.g., checking if the activation value of the neuron \( h_{i,n} \) is less than or equal to 0.3), which gives us the target attribute with values 0 (indicating the condition in the term is not satisfied, e.g., the value is greater than 0.3) and 1 (indicating the condition in the term is satisfied, e.g., the value is less than or equal to 0.3). We give the activation values for layer \( h_{i-1} \) and calculated target attribute on the activation values for layer \( h_i \) to C5.0 as an input, and C5.0 returns a decision tree.

C5.0 has an optional ability to convert a decision tree into a rule set that consists of an unordered collection of simple IF-THEN rules. As described in Section 2.9, rules
are generated as following all paths from the root of the tree to its leaves, and afterwards, they are simplified by pruning. Our algorithm uses C5.0 to build rule sets, instead of trees themselves. Moreover, we take advantage of C5.0’s feature winnow (see Section 2.9) to only use important attributes when creating a decision tree.

The first step of our algorithm is to build the rules, which map the last hidden layer of a neural network to the output layer (Algorithm 1, line 3). In this step only, our algorithm does not calculate the target attribute to pass it to C5.0, as previously described. Instead, it passes on the class values \{λ₁, λ₂, ..., λ_c\} (line 2). The function classWhenMax is defined as: classWhenMax(h_{k+1}(x)) = λ_u \in \{λ₁, λ₂, ..., λ_c\}, where h_{k+1,u}(x) is maximal. So, for the \( j \)-th training example \( x_j \), the algorithm looks at the values of the output neurons for this example, that is \( h_{k+1,1}(x_j), ..., h_{k+1,c}(x_j) \). If the maximum value of \( h_{k+1,1}(x_j), ..., h_{k+1,c}(x_j) \) is \( h_{k+1,u}(x_j) \), then \( λ_u \) is assigned to \( x_j \). The result of the C5.0 call in (line 3) are the rules, mapping the last hidden layer to the output layer: \( R_{h_k \rightarrow o} \).

**Example:** The neural network’s structure is 100-32-8-2, which means that the network consists of two hidden layers. At the beginning, the algorithm uses classification model C5.0 to build a decision tree that maps the activation values of the second hidden layer to the output layer, predicting either ER- or ER+. C5.0 does not use all attributes (see feature winnow in Section 2.9), so even though the network has eight neurons in the second hidden layer, it only used neurons h21 and h23 for building a decision tree (Fig. 9).

![Decision tree, mapping the second hidden layer to the output layer.](image)

**Example:** As we already said, in this toy example only, rules will be generated as simply following paths of the tree from its root to the leaves. Therefore, in the first step, the algorithm generates a rule set \( R_{h_k \rightarrow o} \), in this case \( R_{h_2 \rightarrow o} \), with three intermediate rules:

- IF \( h_{23} \leq 0.3 \) & \( h_{21} \leq 0.7 \) THEN ER-,
- IF \( h_{23} \leq 0.3 \) & \( h_{21} > 0.7 \) THEN ER+,
- IF \( h_{23} > 0.3 \) THEN ER+.

Next, the algorithm extracts unique terms that appear in the rule set \( R_{h_k \rightarrow o} \) (line 6) and for each of them, it builds new rules, which map activation values from the previous layer \( (h_{k-1}) \) to zeros and ones (the process of building rules is described at the beginning of this section). Input data for C5.0 are activation values for each of the training examples on layer \( h_{k-1} \) (line 7). The target values for C5.0 come from applying
the current term \( t \) to the activation values in the hidden layer \( h_k \); if the term holds, it takes one, otherwise zero (line 10). The result of building the rules for each of the terms in \( R_{h_k \rightarrow o} \) is the rule set \( R_{h_{k-1} \rightarrow h_k} \). This process is repeated for each of the hidden layers until we get the rule set \( R_{i \rightarrow h_1} \) that describes the first hidden layer based on the input layer.

**Example:** There are two unique terms in a rule set \( R_{h_2 \rightarrow o} \): \( h_23 \leq 0.3 \) and \( h_21 \leq 0.7 \). For each of them, the algorithm builds new decision trees, which map the first hidden layer to the second hidden layer (Figs. 10a and 10b). Decision trees are built with C5.0, which takes as an input the activation values of the layer \( h_1 \), and target attribute consisting of zeros and ones. For example, let us look at how the decision tree for the term \( h_23 \leq 0.3 \) is built. A part of the input to C5.0 are raw activation values of the layer \( h_1 \). The second part is the target attribute, which is calculated as follows. For all training examples, the algorithm checks if the activation value for a neuron \( h_23 \) is less than or equal to 0.3 or greater than 0.3. If the first condition is satisfied, value for the particular example is 1, otherwise it is 0. Then, C5.0 builds a decision tree, which contains conditions on the neurons of the layer \( h_1 \), and has values 1 (meaning that the condition in a term was satisfied) or 0 in leaves; 1 represents \( h_23 \leq 0.3 \) and 0 represent \( h_23 > 0.3 \). For better understanding, we include whole terms in the leaves, instead of zeros and ones. The returned decision tree is shown in Fig. 10a. The same procedure is done for the other term, that is, \( h_21 \leq 0.7 \). The result is a decision tree, shown in Fig. 10b. A union of rules, representing terms \( h_23 \leq 0.3 \) and \( h_21 \leq 0.7 \), forms a rule set \( R_{h_1 \rightarrow h_2} \). In the next iteration, this process is repeated for expressing the first hidden layer in terms of the neurons in the input layer. This means that for each of the appearing terms in a rule set \( R_{h_1 \rightarrow h_2} \), the algorithm creates new decision trees, which express the applicability of the terms in \( R_{h_1 \rightarrow h_2} \) based on the input features (Figs. 11a, 11b, 11c, and 11d). The union of these rules represent a rule set \( R_{i \rightarrow h_1} \).

![Decision tree for the term h23 ≤ 0.3.](image1)

![Decision tree for the term h21 ≤ 0.7.](image2)

Figure 10: Decision trees, mapping the first hidden layer to the second hidden layer.

After the loop (line 4) is finished, we have \( k+1 \) rule sets, which describe each layer of the neural network (except for the input layer) in terms of the neurons in the preceding layer:

\[
R_{i \rightarrow h_1}, R_{h_1 \rightarrow h_2}, \ldots, R_{h_{k-1} \rightarrow h_k}, R_{h_k \rightarrow o}.
\]

The end goal is to get the rule set \( R_{i \rightarrow o} \) by merging these rule sets.
3.2.2 Merging the rule sets

Just like building, merging \( k + 1 \) rule sets also happens in reverse order. First, rule sets \( R_{h_{k-1} \rightarrow h_k} \) and \( R_{h_k \rightarrow o} \) are merged and the result is the rule set \( R_{h_{k-1} \rightarrow o} \), mapping the second to last hidden layer to the output layer. Next, rule sets \( R_{h_{k-2} \rightarrow h_{k-1}} \) and \( R_{h_{k-1} \rightarrow o} \) are merged, producing the rule set \( R_{h_{k-2} \rightarrow o} \). At each iteration (line 11) we express the output layer with tests on the neurons that are one layer closer to the input until we get to the input layer. In the end, we get the rule set \( R_i \rightarrow o \), which maps the attributes, given to the input layer, to the classification values.

Merging two rule sets \( R_{h_{j-1} \rightarrow h_j} \) and \( R_{h_j \rightarrow o} \) is done as follows. Each appearing term in \( R_{h_j \rightarrow o} \) is replaced by its corresponding rules in a rule set \( R_{h_{j-1} \rightarrow h_j} \). Remember, when the algorithm builds rule sets for two consecutive layers, it does so for every term that appears throughout the builds. Therefore, when we are merging rule sets \( R_{h_{j-1} \rightarrow h_j} \) and \( R_{h_j \rightarrow o} \), each term in \( R_{h_j \rightarrow o} \) will have its own rule subset in \( R_{h_{j-1} \rightarrow h_j} \). The merging is done in such a way, that all possible combinations are looked at. For getting a clearer
idea of what happens when two sets are merged, we show this on the example.

Example: When having a neural network with two hidden layers, two iterations of two loops (Algorithm 1, lines 4 and 11) happen. At the moment, all rule sets that connect the two consecutive layers had been generated. The next step is merging these rule sets in reverse order. After merging rule sets $R_{h_1 \rightarrow h_2}$ and $R_{h_2 \rightarrow o}$, we get the tree, shown in Fig. 12. Therefore, rule set $R_{h_1 \rightarrow o}$ consists of the following rules:

- IF $h_{17} \leq 0.2$ & $h_{18} \leq 0.6$ THEN ER+,
- IF $h_{17} \leq 0.2$ & $h_{18} > 0.6$ & $h_{15} \leq 0.9$ THEN ER-,
- IF $h_{17} \leq 0.2$ & $h_{18} > 0.6$ & $h_{15} > 0.9$ & $h_{121} \leq 0.8$ THEN ER-,
- IF $h_{17} \leq 0.2$ & $h_{18} > 0.6$ & $h_{15} > 0.9$ & $h_{121} > 0.8$ THEN ER+, and
- IF $h_{17} > 0.2$ THEN ER+.

![Figure 12: Decision tree, mapping the first hidden layer to the output layer.](image)

Example: In the last step, rule sets $R_{i \rightarrow h_1}$ and $R_{h_1 \rightarrow o}$ are merged (Fig. 13). The following are the final rules we get after merging. Their conditions are set on the features that are the input to the neural network, and their decision is either ER- or ER+.

1. IF $i_{72} \leq 0.1$ & $i_{33} \leq 0.9$ THEN ER+
2. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} \leq 0.6$ & $i_{87} \leq 0.5$ & $i_{0} \leq 0.6$ THEN ER-
3. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} \leq 0.6$ & $i_{87} \leq 0.5$ & $i_{0} > 0.6$ THEN ER+
4. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} \leq 0.6$ & $i_{87} > 0.5$ & $i_{16} \leq 0.3$ THEN ER+
5. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} \leq 0.6$ & $i_{87} > 0.5$ & $i_{16} > 0.3$ THEN ER-
6. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} > 0.6$ & $i_{25} \leq 0.8$ & $i_{87} \leq 0.5$ & $i_{0} \leq 0.6$ THEN ER-
7. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} > 0.6$ & $i_{25} \leq 0.8$ & $i_{87} \leq 0.5$ & $i_{0} > 0.6$ THEN ER+
8. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} > 0.6$ & $i_{25} > 0.8$ & $i_{87} > 0.5$ & $i_{16} \leq 0.3$ THEN ER+
9. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} > 0.6$ & $i_{25} > 0.8$ & $i_{87} > 0.5$ & $i_{16} > 0.3$ THEN ER-
10. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} \leq 0.7$ & $i_{7} > 0.6$ & $i_{25} > 0.8$ THEN ER-
11. IF $i_{72} \leq 0.1$ & $i_{33} > 0.9$ & $i_{47} > 0.7$ & $i_{42} \leq 0.2$ THEN ER+
12. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} \leq 0.6 \& i_{87} \leq 0.5 \& i_{0} \leq 0.6$ THEN ER-
13. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} \leq 0.6 \& i_{87} \leq 0.5 \& i_{0} > 0.6$ THEN ER+
14. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} \leq 0.6 \& i_{87} > 0.5 \& i_{16} \leq 0.3$ THEN ER+
15. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} \leq 0.6 \& i_{87} > 0.5 \& i_{16} > 0.3$ THEN ER-
16. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} > 0.6 \& i_{25} \leq 0.8 \& i_{87} \leq 0.5 \& i_{0} \leq 0.6$ THEN ER-
17. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} > 0.6 \& i_{25} \leq 0.8 \& i_{87} \leq 0.5 \& i_{0} > 0.6$ THEN ER+
18. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} > 0.6 \& i_{25} \leq 0.8 \& i_{87} > 0.5 \& i_{16} \leq 0.3$ THEN ER+
19. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} > 0.6 \& i_{25} \leq 0.8 \& i_{87} > 0.5 \& i_{16} > 0.3$ THEN ER-
20. IF $i_{72} \leq 0.1 \& i_{33} > 0.9 \& i_{87} > 0.7 \& i_{42} > 0.2 \& i_{7} > 0.6 \& i_{25} > 0.8$ THEN ER-
21. IF $i_{72} > 0.1$ THEN ER+

Let us now take a step back and look at how the way of building and merging rule sets affects the conditions in the rules. Let us assume that the algorithm created all rule sets connecting two consecutive layers, and it already performed some iterations of merging. Without loss of generality, we can assume that in the last iteration, a rule set $R_{h_{j+1} \rightarrow o}$ was generated. Then there exists some rule with conditions on the neurons of the layer $h_{j+1}$, predicting some class from the output layer, for example "IF $h_{j+1,4} \leq 0.8$ AND $h_{j+1,1} > 0.5$ AND $h_{j+1,7} \leq 0.6$ THEN X".

In the next step, the algorithm will merge rule sets $R_{h_{j} \rightarrow h_{j+1}}$ and $R_{h_{j+1} \rightarrow o}$. A part of the merge step will be replacing terms appearing in this example rule, with the corresponding rules in $R_{h_{j} \rightarrow h_{j+1}}$. Let us remember how the corresponding rules were generated. For example, let us look at the term $h_{j+1,4} \leq 0.8$. The algorithm passed the activation values of the layer $h_{j}$ and the computed target attribute (consisting of ones and zeros, depending on whether the condition in the term $h_{j+1,4} \leq 0.8$ was satisfied on activation values of neuron $h_{j+1,4}$ for training examples) to C5.0, which then returned a set of rules, predicting either 0 or 1.

Returned rules consist of the terms with conditions on neurons in the layer $h_{j}$. Their conditions are not affected by how the rules for other terms (e.g., $h_{j+1,1} > 0.5$ and $h_{j+1,7} \leq 0.6$) are built. Therefore, it is possible that some neuron in $h_{j}$, for example $h_{j,2}$, appears in all three rule sets, which represent each of the terms: $h_{j+1,4} \leq 0.8$, $h_{j+1,1} > 0.5$, and $h_{j+1,7} \leq 0.6$.

Let us take a look at the possible relevant subsets of rules for these three terms:

1. $h_{j+1,4} \leq 0.8$:
   - IF $h_{j,2} \leq 0.2$ AND $h_{j,5} \leq 0.3$ THEN 1
2. $h_{j+1,1} > 0.5$:
   - IF $h_{j,8} \leq 0.6$ THEN 1
3. $h_{j+1,7} \leq 0.6$:
   - IF $h_{j,2} \leq 0.3$ AND $h_{j,1} > 0.8$ THEN 1
Figure 13: Decision tree, mapping the input layer to the output layer.
- IF $h_{j,3} > 0.7$ AND $h_{j,2} > 0.9$ THEN 1

These rules all predict value 1, which means that they satisfy conditions in their corresponding terms from layer $h_{j+1}$. During the merge step, these rules will replace their respective terms, resulting in the following rules:

A: IF $h_{j,2} \leq 0.2$ AND $h_{j,5} \leq 0.3$ AND $h_{j,8} \leq 0.6$ AND $h_{j,2} \leq 0.3$ AND $h_{j,1} > 0.8$ THEN X

B: IF $h_{j,2} \leq 0.2$ AND $h_{j,5} \leq 0.3$ AND $h_{j,8} \leq 0.6$ AND $h_{j,3} > 0.7$ AND $h_{j,2} > 0.9$ THEN X

We can see that it can occur that there are two conditions for the same neuron in one rule. A term is called redundant if the rule includes a more specific term. For example, a term $h_{j,2} \leq 0.3$ in rule A is redundant, because there exist more specific term $h_{j,2} \leq 0.2$ in the same rule. We can also get rules that are logically inconsistent, that is, there are two terms in one rule that are mutually exclusive. For example, terms $h_{j,2} \leq 0.2$ and $h_{j,2} > 0.9$ in rule B can not be both true at the same time. Rules with mutually exclusive terms are called unsatisfiable (e.g., rule B). Additionally, after every merging of two rule sets, the algorithm removes redundant terms and intermediate (rules that do not map input features to the network’s output) unsatisfiable rules.

Example: After each merge step, we remove redundant terms and unsatisfiable rules before moving on to the next merging. After the merging of rule sets $R_{h_1 \rightarrow h_2}$ and $R_{h_2 \rightarrow o}$, no redundant terms or unsatisfiable rules arise, but after the second merge, the algorithm removes some redundant terms and unsatisfiable rules. For example, the term $i87 \leq 0.7$ in the second rule is redundant, because there is another more strict condition $i87 \leq 0.5$ in the same rule. After removing redundant terms, the rule can have at most two conditions for the same neuron: one for $\leq$ and one for $>$ (e.g., $i87 \leq 0.7$ and $i87 > 0.5$ in the fourth rule). While removing redundant terms can not decrease the number of rules, removing unsatisfiable rules can. For example, there are two conditions for neuron $i87$ in rule twelve and they can not both be true at the same time, so the algorithm removes this rule. Rules 2, 3, 6, 7, 14, 15, 18, and 19 each contain one redundant term. Rules 12, 13, 16, and 17 are unsatisfiable. There are two terms for the same neuron in the rules 4, 5, 8, and 9, but they are not redundant, nor do they make the rule unsatisfiable. After merging and removing redundant terms and unsatisfiable rules, the number of rules decreases from 21 to 17, and the average number of terms per rule decreases from 6.2 to 5.4. With the greater number of rules, this decrease gets even more noticeable.

### 3.3 Data

For our experiments, we use data from METABRIC (Molecular Taxonomy of Breast Cancer International Consortium) data set, which is a Canada-UK project with the aim to classify breast tumours. Further subcategories help determine the optimal course of treatment and are based on molecular signatures. The studies discovered new genes that are connected to breast cancer. Genes that drive the disease are targets for new drugs that may be developed in the future [29]. The METABRIC data set contains multiple genomic, transcriptomic and image data for 1,980 breast cancer patients. In our study, we include two types of features: gene expressions (mRNA) and copy number alterations (CNA).
Messenger RNA (mRNA) acts as a genetic template since DNA cannot leave the nucleus and is therefore a messenger between DNA and protein production. DNA dosage is often altered in cancer and is a significant determinant of mRNA expression [49].

Copy number alterations (CNA) are somatic changes in the copy numbers of a DNA sequence that appear throughout the cancer development [8]. Investigating these genomic alterations in breast cancer patients can provide essential insights into breast cancer pathogenesis and discover potential biomarkers. Moreover, it can come up with novel drug targets for better therapeutic treatment options [44]. Several studies have found CNAs to be an especially common genetic mutation in cancer [4, 50].

We use METABRIC data set to train our deep neural networks on. We experimented with two classification tasks, for which we built three networks with different architectures. We discuss these networks, together with the information about the implementation, in the following section.

3.4 Experimental setup

We consider two classification tasks. First, we classify for two classes of ER expression. Our second task is a multi-class classification problem with five possible classes of PAM50 subtypes. In what follows, the biological importance of these two targets will be explained.

Estrogen receptor (ER) expression is the most important indicator of possible response to hormonal therapy, and the patient can be either ER positive (ER+) or ER negative (ER-). Cells in the ER+ tumour have receptors that allow them to use the hormone estrogen to grow, so patients with this kind of tumours can be treated with an anti-estrogen therapy that can block the growth of the cancer cells [27]. Prediction analysis of microarray (PAM50) tests a tumour for fifty genes and predicts one of the five subtypes: luminal A (LumA), luminal B (LumB), HER2-enriched, basal-like, and normal-like. These subtypes can be used to evaluate the likelihood of efficacy from neoadjuvant chemotherapy, which is a treatment given as a first step to shrink a tumour before the main treatment. Additionally, PAM50 predicts the risk of relapse (ROR) score. Cancer relapse or recurrence means that cancer comes back after treatment, and after some period of time when cancer could no longer be detected [32].

Our data consists of a preselected set of the mRNA and CNA features used in [12]. Specifically, we use 1,000 features of normalised gene expression numerical data, scaled to an interval [0, 1], and 1,000 features of copy number categorical data. Data is sampled into 5-fold cross validation splits, stratified according to the class distribution in target variables ER and PAM50.

Overall, we ran three different experiments:

- predicting ER with 1,000 mRNA features,
- predicting ER with 1,000 mRNA and 1,000 CNA features, and
- predicting PAM50 with 1,000 mRNA and 1,000 CNA features.

Target variables and used features are partially determining the structure of the neural network for each of the experiments. For the sake of simplicity, we decided to experiment on networks with four layers in total. Let us denote used models as DNN₁,
DNN₂, and DNN₃. The sizes of input and output layers are increasing with each of the structures. DNN₁ has 1,000 inputs and two output neurons and it maps gene expression data (mRNA) to the ER expression. DNN₂ has 2,000 inputs and two output neurons and it maps a combination of CNA and mRNA data to the ER expression. Lastly, DNN₃ has 2,000 inputs and five output neurons and it maps mRNA and CNA data to five PAM50 subtypes.

In general, there can be either one or two output neurons when dealing with the binary classification. As we will later extract rules from the network (see Section 3.2), we require two output neurons, coinciding with two classes.

Besides the input and output sizes that are fixed, we performed a grid-search for other parameters: sizes of two hidden layers (which conclusively determine the network’s structure), the number of epochs and batch size. These parameters are presented in Table 3. For each of the three structures, we tested all configurations, that is, a Cartesian product of the values shown in Table 3. The end goal of manually tuning hyperparameters is optimal performance on the test set [18].

<table>
<thead>
<tr>
<th>Model</th>
<th>1st Hidden Layer</th>
<th>2nd Hidden Layer</th>
<th>Epoch</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN₁</td>
<td>512, 256, 128</td>
<td>64, 32, 16, 8, 4</td>
<td>100, 50</td>
<td>64, 32, 16</td>
</tr>
<tr>
<td>DNN₂, DNN₃</td>
<td>512, 256, 128</td>
<td>64, 32, 16, 8</td>
<td>100, 50</td>
<td>64, 32, 16</td>
</tr>
</tbody>
</table>

Table 3: Grid-search parameters: number of neurons in the first and second hidden layers, number of epochs and batch size.

For the implementation in Python, Keras library [23] was used. We employed a softmax activation function on the output layer and rectified linear units (ReLU), that is, \( f(z) \equiv \max(0, z) \), on other layers. We configured the learning process the following way. We used Adam optimiser [31] with its default values, categorical cross entropy for calculating a loss, and accuracy as a metric. We performed 5-fold cross validation and chose the configuration, which had the highest average accuracy over five folds.

The optimal configurations for each of the models are presented in Table 4, with the accuracy on the test set in the last column. The structure of a model is represented with the numbers of neurons for all layers. For example, DNN₁’s structure is 1000-128-32-2, which means that it has 1,000 input neurons, 128 neurons in the first hidden layer, 32 neurons in the second hidden layer, and two output neurons.

<table>
<thead>
<tr>
<th>Model</th>
<th>Input Data</th>
<th>Output Data</th>
<th>Structure</th>
<th>Epoch</th>
<th>Batch</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN₁</td>
<td>mRNA</td>
<td>ER</td>
<td>1000-128-32-2</td>
<td>50</td>
<td>64</td>
<td>96.31%</td>
</tr>
<tr>
<td>DNN₂</td>
<td>mRNA, CNA</td>
<td>ER</td>
<td>2000-256-16-2</td>
<td>50</td>
<td>32</td>
<td>95.91%</td>
</tr>
<tr>
<td>DNN₃</td>
<td>mRNA, CNA</td>
<td>PAM</td>
<td>2000-256-32-5</td>
<td>100</td>
<td>64</td>
<td>75.99%</td>
</tr>
</tbody>
</table>

Table 4: Chosen parameters.

One part of the rule extraction is implemented in R. In particular, we used package C5.0 [6] to build rules for two consecutive layers of the neural network (see Section 3.2).

We have described the setup for our experimental part. We are familiar with the tasks that will be performed by neural networks, from which we will extract rules. The extracted set of rules will describe the neural network’s output layer with input features.
These rules approximate the behaviour of the neural network, and their predictions will sometimes differ. We talk about measures used to evaluate the extracted rules in the following section.

### 3.5 Evaluation measures

Extracted rules approximate deep neural networks, which is why their predictions will sometimes differ. We need to evaluate how good the approximation is. Andrews et al. [2] define measures for evaluating rules, extracted from neural networks as follows.

**Accuracy** A rule set is called accurate if it can correctly classify unseen examples.

**Fidelity** A rule set has a high level of fidelity if it can mimic the behaviour of the neural network, from which it was extracted.

**Comprehensibility** The comprehensibility of a rule set is dependent on the number of rules in a rule set and the number of conditions per rule.

We evaluate the results with respect to seven measures: time, memory, accuracy (comparison between the rule set’s predictions with the correct class), fidelity (comparison between the rule set’s prediction and the prediction of a neural network), rule set size, average rule length and features, extracted from rules. We evaluate the deep neural networks for accuracy, and then the extracted rules for all seven measures.

Measured time represents how long did the algorithm require to extract rules from the corresponding neural network. For measuring memory, Python’s library Guppy 3 [19] is used. We report the total size of all created objects, which are generated while extracting rules from a neural network.

When making a prediction for a previously unseen example, all applicable rules play a role. A rule is applicable if all of its conditions are fulfilled. The prediction is then made as a majority vote of all applicable rules. If none of the rules in a rule set is applicable, then a default class is predicted. The default class for a rule set is calculated from the frequency statistics of samples in the training set that are not classified using rules. The most frequent class among these examples is selected as a default prediction.

The evaluation of the rule set $R$ is done on the testing data $x_1, x_2, \ldots, x_t$, where each $x_j$ has a corresponding true class $y_j \in \{\lambda_1, \lambda_2, \ldots, \lambda_c\}$. We look at the three values for each example $x_j$:

- example’s true class $y_j$,
- prediction of a neural network, which was used for rule extraction, $d_j$, and
- prediction of the rule set $R$, that is, $c_j$.

We denote the number of correctly predicted examples by rule set $R$ as $N_{rs, true}$. This means that for $N_{rs, true}$ examples $c_j$ equals $y_j$. Similarly, $N_{rs, nn}$ is the number of examples, where predicted classes by a rule set coincides with the predicted class by a neural
network, from which the rule set was extracted, that is, $c_j$ equals $d_j$. Then we define accuracy and fidelity of the rule set $R$ as:

$$accuracy(R) = \frac{N_{rs,\text{true}}}{t}$$ (3.1)

$$fidelity(R) = \frac{N_{rs,\text{nn}}}{t}$$ (3.2)

In other words, the accuracy of a rule set is defined as the percentage of correctly classified examples, and its fidelity as the percentage of examples classified the same as with neural network. Therefore, fidelity tells us how well does a rule set mimic the neural network, from which it was extracted.

There is no exact formula for calculating comprehensibility measure, but comparing two rule sets, the smaller rule set with shorter rules is more comprehensible than the other one. So, we measure comprehensibility through two measures: rule set size and average rule length. Rule of the form

$$\text{IF test}_1 \text{ AND test}_2 \text{ AND } \ldots \text{ AND test}_L \text{ THEN class = X}$$

has length $L$. If the rule set $R$ contains $N$ rules with respective lengths $L_1, L_2, \ldots, L_N$, we calculate the average rule length as:

$$\text{avg rule length}(R) = \frac{\sum_{i=1}^{N} L_i}{N}$$ (3.3)

The last measure, that is, features, extracted from rules, does not have the explicit definition. We use it to give us some more insight into what features are very likely to be important. Nevertheless, the main goal of our rule extraction algorithm is to explain deep neural networks. Therefore, we compare attributes that appear in rule sets provided by different algorithms (i.e., our algorithm and two baselines), and look at the ones that appear multiple times.

Table 5 presents a simplified example of a rule set (with ten rules) we get after the rule extraction. The features used are gene expressions (GE) for different genes (e.g., GSDMB and WWP1). Each row represents one rule with its conditions listed in the second column and predicted class in the third column. They predict estrogen receptor expression, which can be either ER- or ER+.

For the evaluation, we split our data into two sets: the training set comprises 80% of the data, while the remaining 20% is included in the test set. When calculating accuracy and fidelity, we make predictions for unseen examples from the test set. For example, consider a test example with values for gene expression features, provided in Table 6. What would the rule set from Table 5 predict for this particular example? First, all applicable rules are identified, that is, 1, 3, 4, 8, and 9. Four of these rules predict ER+, while only one predicts ER-. We look at the majority vote and get the final prediction ER+. 
<table>
<thead>
<tr>
<th>Rule</th>
<th>Conditions</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GE_GSDMB \leq 0.46 &amp; GE_DHTKD1 \leq 0.42 &amp; GE_NME3 &gt; 0.58 &amp; GE_WWP1 &gt; 0.27</td>
<td>ER+</td>
</tr>
<tr>
<td>2</td>
<td>GE_DHTKD1 \leq 0.63 &amp; GE_NME3 &gt; 0.58 &amp; GE_PSMC3IP &gt; 0.31 &amp; GE_WWP1 &gt; 0.27</td>
<td>ER-</td>
</tr>
<tr>
<td>3</td>
<td>GE_CCNE1 \leq 0.14 &amp; GE_WWP1 &gt; 0.27</td>
<td>ER+</td>
</tr>
<tr>
<td>4</td>
<td>GE_DHTKD1 \leq 0.63 &amp; GE_NME3 &gt; 0.68 &amp; GE_WWP1 &gt; 0.27</td>
<td>ER+</td>
</tr>
<tr>
<td>5</td>
<td>GE_PSMC3IP &gt; 0.28 &amp; GE_WWP1 &gt; 0.38</td>
<td>ER-</td>
</tr>
<tr>
<td>6</td>
<td>GE_ERLIN2 &gt; 0.45 &amp; GE_KCTD3 &gt; 0.46</td>
<td>ER+</td>
</tr>
<tr>
<td>7</td>
<td>GE_WWP1 \leq 0.38 &amp; GE_MTL5 \leq 0.20 &amp; GE_ABAT \leq 0.23</td>
<td>ER-</td>
</tr>
<tr>
<td>8</td>
<td>GE_WWP1 \leq 0.38 &amp; GE_ABAT \leq 0.23 &amp; GE_RARA \leq 0.47</td>
<td>ER-</td>
</tr>
<tr>
<td>9</td>
<td>GE_WWP1 \leq 0.38 &amp; GE_PNMT &gt; 0.40 &amp; GE_ABAT \leq 0.23</td>
<td>ER+</td>
</tr>
<tr>
<td>10</td>
<td>GE_WWP1 \leq 0.38 &amp; GE_MTL5 \leq 0.16 &amp; GE_ADRM1 &gt; 0.22</td>
<td>ER-</td>
</tr>
</tbody>
</table>

Table 5: Example of a rule set from METABRIC dataset.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE_GSDMB</td>
<td>0.23</td>
</tr>
<tr>
<td>GE_DHTKD1</td>
<td>0.11</td>
</tr>
<tr>
<td>GE_NME3</td>
<td>0.82</td>
</tr>
<tr>
<td>GE_WWP1</td>
<td>0.31</td>
</tr>
<tr>
<td>GE_PSMC3IP</td>
<td>0.16</td>
</tr>
<tr>
<td>GE_CCNE1</td>
<td>0.07</td>
</tr>
<tr>
<td>GE_ERLIN2</td>
<td>0.55</td>
</tr>
<tr>
<td>GE_KCTD3</td>
<td>0.09</td>
</tr>
<tr>
<td>GE_MTL5</td>
<td>0.79</td>
</tr>
<tr>
<td>GE_ABAT</td>
<td>0.14</td>
</tr>
<tr>
<td>GE_RARA</td>
<td>0.36</td>
</tr>
<tr>
<td>GE_PNMT</td>
<td>0.43</td>
</tr>
<tr>
<td>GE_ADRM1</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Table 6: Input values for one specific test example.

In Chapter 3, we first described the design of our experimental part. Then, Section 3.2 gave a detailed description of our rule extraction algorithm of the decompositional nature, together with the running example to illustrate the functionality of the algorithm. We familiarised ourselves with the data used for deep neural networks to train on. Next, we discussed the classification tasks, we deal with, and described three different experiments that we performed. The result of our algorithm is a set of rules, mapping the input features to the output layer of the network. Lastly, we defined measures for evaluating extracted rules.
4 Experiments and evaluation

In this chapter, we present the results of using our rule extraction algorithm for the tasks described in Section 3.4. The extracted rules are evaluated with the measures defined in Section 3.5. We compare our algorithm with two baselines, which we discuss below. The main goal of the experiments is to show that our algorithm creates better rules than existing decompositional algorithms, able to extract rules from deep neural networks. With better, we refer to smaller rule sets with shorter rules and higher accuracy. We also show that rule sets generated with our algorithm mimic the behaviour of the neural network better. With comparing our algorithm to the pedagogical baseline, we want to show that investing more time and memory resources pays off with the better approximation of the network's behaviour.

After choosing the hyperparameters for each of the three models $DNN_1$, $DNN_2$, and $DNN_3$ in Section 3.4, and training the corresponding models, we extract rules from them. Our algorithm has a decompositional approach to rule extraction, which means that it takes into consideration the network's whole structure, that is, input and output layers, as well as hidden layers. It generates extracted rules based on the activation values of all neurons. In comparison with pedagogical algorithms, it requires additional computational time and memory.

For the sake of comparison, we include a baseline of each variety. As the first baseline, we implemented a so-called pedagogical C5.0, which builds rules with C5.0 (described in Section 2.9) on the raw data (input features) and the output of the neural network. The second baseline is a decompositional algorithm DeepRED (described in Section 2.7). To our knowledge, DeepRED is the only decompositional algorithm that has been explicitly tested on the task of extracting rules from deep neural networks. One of the major differences between our algorithm and DeepRED is that DeepRED extracts rules for each of the classes separately, while our algorithm extracts rules for all classes at once. We evaluate rule sets generated with our algorithm and with the two baselines with measures, described and defined in Section 3.5.

Initial experiments The initial experiments showed that the initialisation of the neural network's weights plays a significant role in the number of extracted rules. Therefore, we wanted to find the best initial weights with the following process. First, we fixed the training set, containing 75% of the data. Then, on the same training set, we built five neural networks with different random initial weights. We evaluated each network's accuracy on the remaining 25% of the data. This value did not vary a lot across networks, so we additionally checked the rule sets' sizes after applying our rule extraction algorithm on these networks.

We chose the network with the smallest rule set size, and used its initial weights for further experiments across all five folds. In other words, we fixed the network's initial weights, and extracted rules using different data (across five folds). Unexpectedly, the final result in terms of accuracy of the extracted rule sets was worse than the one with the random weights, so we report results with random weight initialisation. Training set comprises 80% of the data, while the remaining 20% is included in the test set. Additionally, data is stratified according to the class distribution in target variables. Reported accuracy and fidelity are calculated on the test set.
We report the accuracy for rule sets, extracted from networks with three algorithms, that is, our rule extraction algorithm, pedagogical C5.0, and DeepRED. In the results tables, the two baselines are referred to as ped C5.0 and DR for pedagogical C5.0 and DeepRED, respectively. We also report the accuracy of the neural network, from which the rules were extracted. We wanted to use the same network's activation values for evaluating our algorithm and DeepRED, which also uses activation values for building rules. First, we trained corresponding neural networks and then used their activation values to evaluate our algorithm. Afterwards, we wanted to use the same activation values for evaluating DeepRED. Unfortunately, due to DeepRED's old implementation [13], we were not able to use corresponding activation values as an input to generate rule sets with DeepRED.

Because of that, we could not use only one neural network for each of the folds of a particular model. If we take model $DNN_1$ as an example: for each of the five folds, we build and train a neural network with 1,000 input neurons and two output neurons (see Fig. 8) twice, that is, once to extract rules with our algorithm and once to extract rules with DeepRED. Therefore, we report the two values for the accuracy of a neural network, from which the rules are extracted. $NN$ represents the accuracy of a network, from which our algorithm and pedagogical C5.0 extracted rules, while $NN_{DR}$ reports the accuracy of a network, from which DeepRED extracted rules. However, we made an effort to use almost identical neural networks for generating activation values: networks' structures were the same along with all their settings, and they were trained and tested on the same data.

Evaluation of DeepRED Zilke et al. [52] evaluated DeepRED on five different data sets. MNIST data set was the one with the greatest number of attributes, that is, 784. They also limited themselves to problems with two output classes only. They adjusted the C4.5 call to build decision trees of the maximum depth of ten. For making the comparison between DeepRED and our rule extraction algorithm, we have used their code, available from [13]. We left the restriction for decision trees' maximum depth as it was, set to ten. This was due to the significantly longer evaluation of their code otherwise. Also, their code allows the use of only relevant neurons when building decision trees for two consecutive layers. Without this option, running their code was significantly slower, so we chose to experiment with using only relevant neurons. In DeepRED [52], the authors limited each experiment to 24 hours, while we left them running for at most one week, that is, 168 hours.

Furthermore, evaluating DeepRED on the METABRIC data set for the model $DNN_1$ (defined in Section 3.4), which has the least number of input features (i.e., 1,000) and output classes (i.e., 2), took multiple weeks. Even with optimisations for faster evaluation, considering only relevant neurons, three out of ten (five folds, each having two classes) experiments ran out of time. Therefore, we decided for $DNN_2$ and $DNN_3$ to only compare our rule extraction algorithm with the other baseline, that is, pedagogical C5.0.

Because DeepRED extracts rules for each of the classes separately, we report their results as an average over all classes. In practice, this means that we report the average over two classes (the experiments we have done for DeepRED for $DNN_1$ have two classes), or we report one value for one of the classes when the experiment for the other class ran out of time.

42
**Results** We intended to give results for each of the models $DNN_1$, $DNN_2$ and $DNN_3$ separately. They are increasing in the complexity of the networks’ structures, that is, the number of input features increases from 1,000 to 2,000 and number of outputs increases from two to five (see Fig. 8). Because of the DeepRED’s poor performance on the first model, we did not evaluate it on the models $DNN_2$ and $DNN_3$.

Results for the model $DNN_1$ confirm the fact that pedagogical algorithms outperform decompositional algorithms in terms of time, memory, size of the rule set and average rule length. Therefore, it makes no sense to compare our algorithm on models $DNN_2$ and $DNN_3$ only to the pedagogical baseline. Therefore, we present a complete comparison of our algorithm with the two baselines on the model $DNN_1$ in the following section, and give incomplete results for other models in Appendix A.

### 4.1 Mapping mRNA to ER expressions

Model $DNN_1$ describes the neural network that maps 1,000 mRNA features to the ER expression with two possible values: ER- and ER+. The best average accuracy (96.31%) over five folds was achieved with the network’s structure 1000-128-32-2 (i.e., 1000 input neurons, 128 neurons in the first hidden layer, 32 neurons in the second hidden layer, and two output neurons), 50 epochs (number of passes of all training data to the neural network during training), and batch size 64 (number of training examples passed to the neural network at once).

Table 7 shows the accuracy and fidelity of the rule sets generated with our algorithm and two baselines, along with the accuracy of the neural networks, whose activation values were used for rule extraction. Rule sets extracted with our rule extraction algorithm are on average more accurate than those extracted with either of the two baselines. They also better represent the behaviour of the neural network, namely, our algorithm’s classifications differ from those of the network in only 4.45% of the cases on average (see fidelity).

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy [%]</th>
<th>Fidelity [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NN</td>
<td>NN$_{DR}$</td>
</tr>
<tr>
<td>1</td>
<td>96.97</td>
<td>94.19</td>
</tr>
<tr>
<td>2</td>
<td>94.70</td>
<td>90.28</td>
</tr>
<tr>
<td>3</td>
<td>95.20</td>
<td>90.15</td>
</tr>
<tr>
<td>4</td>
<td>96.46</td>
<td>93.18</td>
</tr>
<tr>
<td>5</td>
<td>94.95</td>
<td>88.13</td>
</tr>
<tr>
<td>Average</td>
<td>95.66</td>
<td>91.19</td>
</tr>
</tbody>
</table>

Table 7: Accuracy and fidelity for model $DNN_1$.

Table 8 illustrates the comprehensibility through the rule set size and the average number of terms per rule. Compared with our algorithm, DeepRED extracts fewer rules, but the rules are on average about 50% longer than rules extracted with our algorithm\(^3\). Pedagogical C5.0 generates the most comprehensible rules. On average, they

\(^3\)Note that DeepRED was not able to extract three out of ten rule sets. We can logically assume that if those rule sets were extracted, they would be significantly bigger. Therefore, these results are a bit
are better in terms of accuracy and fidelity than DeepRED's rules but worse than rules extracted with our algorithm. The number of rules extracted with our algorithm and DeepRED varies considerably, for which we have no certain explanations. The unusually large numbers of rules are built for the same fold (i.e., four) with both algorithms, so the number of rules may depend on the data only. There is no clear correlation between the number of generated rules and their accuracy or fidelity.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Rule set size</th>
<th>Average rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ped C5.0</td>
<td>DR</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
<td>5,569</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>7,419</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>2,083</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>42,406</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>403</td>
</tr>
<tr>
<td>Average</td>
<td>19.8</td>
<td>11,576</td>
</tr>
</tbody>
</table>

Table 8: Comprehensibility for model $DNN_j$.

Table 9 encompasses required time and memory (in megabytes). As expected, the pedagogical baseline is at least time and memory demanding. It extracts rules in a matter of seconds, compared to 2.5 minutes, needed by our algorithm. On the other hand, DeepRED is very time and memory expensive. On average, it runs almost 38 hours to extract rules from one network. Three out of ten experiments for DeepRED exceeded the time limit of one week. Our time limit for DeepRED experiments is very generous. If we followed authors of DeepRED and set the time limit to 24 hours, only one in ten runs would extract rules in that time.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Time</th>
<th>Memory [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ped C5.0</td>
<td>DR</td>
</tr>
<tr>
<td>1</td>
<td>0.19</td>
<td>29.95</td>
</tr>
<tr>
<td>2</td>
<td>0.16</td>
<td>27.16</td>
</tr>
<tr>
<td>3</td>
<td>0.17</td>
<td>31.58</td>
</tr>
<tr>
<td>4</td>
<td>0.17</td>
<td>79.70</td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td>20.13</td>
</tr>
<tr>
<td>Average</td>
<td>0.17</td>
<td>37.7</td>
</tr>
</tbody>
</table>

Table 9: Time and memory consumption for model $DNN_j$.

Overall, our rule extraction algorithm is much more time and memory efficient compared to DeepRED. It also provides shorter and more accurate rules, which mimic the behaviour of the neural network better (i.e., they have higher fidelity). On average, DeepRED generates smaller rule sets, but if we take into account that three out of ten rule sets would be likely very large if successfully extracted, our algorithm generates smaller rule sets.

4Because of the significant difference in execution times across the three algorithms, time for pedagogical C5.0 and our algorithm is reported in minutes, while time for DeepRED is reported in hours.

skewed in DeepRED's favour.
The significant variance across the number of rules made us think about what this means in practice. When predicting a target value for a new patient, we most likely do not use the majority of the rules. Only the applicable rules play a part in decision making. We analysed how many rules are applicable for making a prediction on average. The results are presented in Table 10. Values in the first column tell us that all examples have at least one applicable rule, and therefore, a default rule was never used. On average, a very small part of the rule set was used for making the prediction for a particular patient. In some folds, the number of never applicable rules is very high, which is especially outstanding for the fold with a very large rule set (i.e., fold 4). As these results put a new perspective on large rule sets in some cases, we think that this data could be a part of the comprehensibility measure.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Minimum</th>
<th>Median</th>
<th>Average</th>
<th>Maximum</th>
<th>Never applicable</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>201</td>
<td>262</td>
<td>1,216</td>
<td>2660</td>
<td>6,227</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>32</td>
<td>36</td>
<td>114</td>
<td>2</td>
<td>287</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>34</td>
<td>41</td>
<td>136</td>
<td>395</td>
<td>791</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>855</td>
<td>1,369</td>
<td>8,448</td>
<td>231,551</td>
<td>283,520</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>30</td>
<td>34</td>
<td>95</td>
<td>78</td>
<td>435</td>
</tr>
<tr>
<td>Average</td>
<td>1.4</td>
<td>230.4</td>
<td>348.4</td>
<td>2,001.8</td>
<td>46,937</td>
<td>58,252</td>
</tr>
</tbody>
</table>

Table 10: Minimum, median, average, and maximum number of applicable rules, compared to the number of never applicable rules and all rules that are generated with our algorithm for model DNN1.

4.2 Comparison of our algorithm and DeepRED

Because our algorithm and the baseline DeepRED are similar algorithms with the same decompositional approach, we are mostly interested in their comparison. DeepRED extracted only seven out of ten rule sets for model DNN1 in a very liberal time limit. We can logically assume that if those rule sets were extracted, they would be significantly bigger than successfully extracted ones. Furthermore, we can assume that the results would be worse for more complex models (i.e., DNN2 and DNN3), in terms of more computational power and bigger rule sets with longer rules (as it is seen with our algorithm). Therefore, we think it is sensible to compare the comprehensibility of rules extracted with our algorithm for models DNN1 and DNN2 with the comprehensibility of the rule sets extracted with DeepRED for model DNN1. Both models classify for two classes of ER expression, but the second model has twice as many inputs as the first model. This comparison is presented in Table 11.

Model DNN2 also contains outliers across the rule set size, although not on the same folds as model DNN1. Rules produced with our algorithm for the model DNN2 (with 2,000 input features) are on average shorter than the rules generated with DeepRED for the model DNN1 (with 1,000 input features). There has to be done more research on why there are such significant anomalies in the rule set size and how to approach this problem. However, it is promising that even the smallest rule set for model DNN2 with only 75 rules still has quite high accuracy of 90.15% and fidelity of 92.68% (see Appendix A).
In this chapter, we evaluated our rule extraction algorithm. We presented results for our algorithm and two baselines, one of the pedagogical and the other of the decompositional approach, that is, pedagogical C5.0 and DeepRED, respectively. We expected decompositional algorithms to be more time and memory expensive. It turned out that the decompositional baseline was so time demanding, we could not afford to run all experiments with it. This resulted in some missing values. We made a complete comparison of all three algorithms for model $DNN_1$ in Section 4.1 and provided the incomplete comparison for models $DNN_2$ and $DNN_3$ in Appendix A.

As expected, the pedagogical baseline outperformed both decompositional algorithms in time and memory complexity, number of extracted rules and their average length. However, our algorithm provided more accurate rules with a higher level of fidelity. Our algorithm also outperformed the decompositional baseline DeepRED in all perspectives.

<table>
<thead>
<tr>
<th>Fold</th>
<th>$DR_{DNN_1}$</th>
<th>$our_{DNN_1}$</th>
<th>$our_{DNN_2}$</th>
<th>$DR_{DNN_1}$</th>
<th>$our_{DNN_1}$</th>
<th>$our_{DNN_2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5,569</td>
<td>6,227</td>
<td>4,088,725</td>
<td>11.45</td>
<td>8.88</td>
<td>15.3</td>
</tr>
<tr>
<td>2</td>
<td>7,419</td>
<td>287</td>
<td>293</td>
<td>12.9</td>
<td>5.05</td>
<td>5.11</td>
</tr>
<tr>
<td>3</td>
<td>2,083</td>
<td>791</td>
<td>47,886</td>
<td>10.97</td>
<td>6.82</td>
<td>12.37</td>
</tr>
<tr>
<td>4</td>
<td>42,406</td>
<td>283,520</td>
<td>75</td>
<td>14.25</td>
<td>13.75</td>
<td>4.83</td>
</tr>
<tr>
<td>5</td>
<td>403</td>
<td>435</td>
<td>294</td>
<td>2.94</td>
<td>5.98</td>
<td>5.46</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td><strong>58,252</strong></td>
<td><strong>827,454.6</strong></td>
<td><strong>11.59</strong></td>
<td><strong>8.10</strong></td>
<td><strong>8.61</strong></td>
</tr>
</tbody>
</table>

Table 11: Comparison of the comprehensibility of our algorithm (for models $DNN_1$ and $DNN_2$) and DeepRED (for model $DNN_1$).
5 Analysis

This chapter first provides the theoretical complexity of our rule extraction algorithm (described in Section 3.2). Note that the worst-case complexity is presented, which does not take into account the algorithm’s optimisation of using only important neurons when building rule sets with the classification algorithm C5.0 (described in Section 2.9). Because there is no way of knowing how many neurons will be ignored in this process, we must consider all neurons in the analysis.

Next, Section 5.2 presents graphs, which compare our algorithm with two baselines, that is, pedagogical C5.0 and DeepRED (described in Section 4). The goal of this section is to give a visual representation of the algorithm’s performance. We compare results for model DNN₁, because this is the only model, which has available results for all three algorithms. We show that our algorithm outperforms the pedagogical baseline in terms of fidelity, which means that it better mimics the behaviour of the neural network. It also generates more accurate rules with higher fidelity than the decompositional baseline DeepRED.

5.1 Theoretical complexity

There exist two ways to estimate the algorithm’s complexity. The first one is called benchmarking, that is, measuring the algorithm’s memory consumption in bytes and speed in seconds [37]. We already performed benchmarking and described the results in Section 4. The downside of this approach is that it gives us insight into the complexity of an algorithm for a specific program written in a specific language, running on a specific computer, with a specific compiler and specific input data. Therefore, it can be difficult to generalise the results. The second approach is through a mathematical analysis, which is independent of the implementation [37]. In this section, we look at the worst-case scenario. Table 12 shows the meaning of the variables that are used in the analysis.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>number of hidden layers</td>
</tr>
<tr>
<td>h₀, h₁, ..., hₖ+₁</td>
<td>neural network’s layers</td>
</tr>
<tr>
<td>n₀, n₁, ..., nₖ+₁</td>
<td>number of neurons per layer</td>
</tr>
</tbody>
</table>

Table 12: Meaning of the variables.

The algorithm Algorithm 1 on page 27 consists of two main parts: building the rule sets for every two consecutive layers and merging these rule sets to get final rules that map the input to the output layer. We analyse them separately in the following two sections.

5.1.1 Building the rules

The most expensive operation in the first part is a C5.0 call that builds the rules, so this is what we are interested in. How many C5.0 calls does the algorithm have to execute? First, there is one C5.0 call that builds a rule set $R_{h_k \rightarrow o}$. However, the majority of C5.0
calls come from the first loop (Algorithm 1, line 4, on page 27), which is dependent on the number of layers and the number of terms in the rule set from the previous iteration. Fig. 14 illustrates the situation after all rule sets connecting two consecutive layers are built. The maximum number of C5.0 calls to build these rule sets, along with their maximum number of rules and appearing terms are collected in Table 13.

The first C5.0 call (before the loop in line 4) produces a rule set \( R_{h_k \rightarrow o} \), where the maximum number of rules is \( 2^{n_k} \) (i.e., when the tree is fully grown) and the maximum number of terms is \( 2^{n_k} - 1 \) (i.e., when each internal node is unique). In the first iteration of the loop, there can be at most \( 2^{n_k} - 1 \) C5.0 calls, one for each of the terms appearing in a rule set \( R_{h_k \rightarrow o} \). One C5.0 call can return at most \( 2^{n_k} - 1 \) rules with at most \( 2^{n_k} - 1 \) terms. If we multiply this with the number of C5.0 calls that we execute in this step, we get the maximum number of rules/terms, appearing in a rule set \( R_{h_{k-1} \rightarrow h_k} \). If we continue this calculation towards the input layer, we get results, presented in Table 13. Note that the number of C5.0 calls to get a current rule set \( R_{h_j \rightarrow h_{j+1}} \) is equal to the number of terms (non-leaf nodes) in a previous rule set \( R_{h_{j+1} \rightarrow h_{j+2}} \). Overall, the number of C5.0 calls to build the rule sets for all pairs of consecutive layers is:

\[
1 + (2^{n_k} - 1) + \left( \frac{2^{n_k} - 1}{(2^{n_k} - 1 - 1)} \right) + \frac{(2^{n_k} - 1)(2^{n_k-1} - 1)}{(2^{n_k-1} - 1 - 1)} + \cdots + \frac{(2^{n_k} - 1)(2^{n_k-1} - 1)(2^{n_k-2} - 1)}{(2^{n_k-2} - 1 - 1)} + \frac{(2^{n_k} - 1)(2^{n_k-1} - 1)(2^{n_k-2} - 1)(2^{n_k-3} - 1)}{(2^{n_k-3} - 1 - 1)} + \cdots + \frac{(2^{n_k} - 1)(2^{n_k-1} - 1)(2^{n_k-2} - 1)(2^{n_k-3} - 1)\cdots(2^{n_1} - 1)}{(2^{n_1} - 1 - 1)}
\]

\[
= 1 + \sum_{j=1}^{k} \prod_{l=0}^{j-1} (2^{n_k-l} - 1)
\]

(5.1)
<table>
<thead>
<tr>
<th>Rule set</th>
<th>Max. number of rules</th>
<th>Max. number of terms</th>
<th>Max. number of C5.0 calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{h_k \rightarrow o}$</td>
<td>$2^{n_k}$</td>
<td>$2^{n_k} - 1$</td>
<td>1</td>
</tr>
<tr>
<td>$R_{h_{k-1} \rightarrow h_k}$</td>
<td>$(2^{n_k} - 1) \times 2^{n_{k-1}}$</td>
<td>$(2^{n_k} - 1) \times (2^{n_{k-1}} - 1)$</td>
<td>$2^{n_k} - 1$</td>
</tr>
<tr>
<td>$R_{h_{k-2} \rightarrow h_{k-1}}$</td>
<td>$(2^{n_k} - 1) \times (2^{n_{k-1}} - 1) \times 2^{n_{k-2}}$</td>
<td>$(2^{n_k} - 1) \times (2^{n_{k-1}} - 1) \times (2^{n_{k-2}} - 1)$</td>
<td>$(2^{n_k} - 1) \times (2^{n_{k-1}} - 1)$</td>
</tr>
<tr>
<td>$R_{h_{j-1} \rightarrow h_j}$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1) \times 2^{n_j}$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1) \times (2^{n_j} - 1)$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1)$</td>
</tr>
<tr>
<td>$R_{h_{j-2} \rightarrow h_{j-1}}$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1) \times 2^{n_{j-2}}$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1) \times (2^{n_{j-1}} - 1)$</td>
<td>$(2^{n_k} - 1) \times \cdots \times (2^{n_{j+1}} - 1)$</td>
</tr>
</tbody>
</table>

Table 13: The maximum numbers of rules and terms per rule set for two consecutive layers, and the maximum number of C5.0 calls to build the corresponding rule sets.
The number of C5.0 calls depends on the number of layers in a network and is exponentially dependent on the number of neurons per layer.

5.1.2 Merging the rule sets

For a neural network with \( k \) hidden layers, the algorithm builds \( k + 1 \) rule sets describing layers in terms of the neurons in the preceding layer:

\[
R_{i \rightarrow h_1}, R_{h_1 \rightarrow h_2}, \ldots, R_{h_{k-1} \rightarrow h_k}, R_{h_k \rightarrow o}.
\]

We calculated the maximum number of rules that may appear in these rule sets and they are shown in Table 13. In the second part, the algorithm merges these rule sets in a layer-wise manner in reverse order (from the output layer towards the input layer). First, the rule sets \( R_{h_{k-1} \rightarrow h_k} \) and \( R_{h_k \rightarrow o} \) are merged and the result is the rule set \( R_{h_{k-1} \rightarrow o} \). Next, rule sets \( R_{h_{k-2} \rightarrow h_{k-1}} \) and \( R_{h_{k-1} \rightarrow o} \) are merged, producing the rule set \( R_{h_{k-2} \rightarrow o} \). At each iteration, the algorithm expresses the output layer with tests on the neurons that are one layer closer to the input until we get to the input layer. At the end, we get the rule set \( R_{i \rightarrow o} \), which maps the input layer to the output layer.

The merging of rules is done in \( k \) steps. In the \( i \)-th step, the algorithm produces a rule set \( R_{h_{k-i} \rightarrow o} \) by merging the rule sets \( R_{h_{k-i} \rightarrow h_{k-i+1}} \) and \( R_{h_{k-i+1} \rightarrow o} \). The merging is done as illustrated in Fig. 15.

For every rule \( r \) in \( R_{h_{k-i+1} \rightarrow o} \):

For every test \( t \) in \( r \):

Replace \( t \) with all corresponding rules from \( R_{h_{k-i} \rightarrow h_{k-i+1}} \).

Figure 15: Pseudo-code for the \( i \)-th merge step.

The maximum number of rules is exponentially dependent on the number of neurons per network layer, so in the worst-case scenario, the number of rules is exponentially growing with each merge step. The worst-case analysis does not look promising at first sight, but because of the way the decompositional algorithm for rule extraction operates, this is expected. We improved this complexity with the use of only important attributes when building the rule sets for two consecutive layers, but because this is solely dependent on the data used, an analysis of the improved algorithm is not possible.

Even if the theoretical complexity of our algorithm is exponential, we show in the next section that the algorithm’s performance is still better than that of DeepRED, which is used as a decompositional baseline. The authors of DeepRED did not explicitly analyse the complexity of their algorithm, so we are not able to make a direct comparison. But since our algorithm and DeepRED have a similar approach to rule extraction, their complexities are similar. The most obvious difference between the algorithms is that DeepRED extracts rules for each of the classes separately, so it is additionally dependent on the size of the output layer, which our algorithm is not. An in-depth analysis should be performed to more accurately compare the complexity of the algorithms.
5.2 Comparison with two baselines

We presented results of evaluating our algorithm and two baselines (i.e., pedagogical C5.0 and DeepRED) on different tasks (defined in Section 3.4) in Chapter 4. In this chapter, we make an overall comparison of the three algorithms and analyse the performance of our algorithm. We show that the pedagogical baseline creates more comprehensible rules (i.e., smaller rule sets with shorter rules) than either of the decompositional algorithms. However, our algorithm generates rule sets with higher fidelity compared to those generated with pedagogical C5.0. Comparing our algorithm to the decompositional baseline, which was not able to produce all rule sets even for the smallest model, our algorithm outperforms it.

It is only logical that pedagogical algorithms are faster and the difference is even more noticeable with deep neural networks, where there are more layers that decompositional algorithms additionally have to process. We are interested to see if putting more time into the rule extraction with decompositional algorithms produces more accurate rules with higher fidelity, compared to pedagogical algorithms as well as compared to another decompositional baseline DeepRED.

Fig. 16 illustrates the needed time and memory to extract rules with the three algorithms. Due to the large differences between the algorithms, both time and memory are displayed on a logarithmic scale. But even this way, we can not pinpoint all 17 runs (i.e., five for each, pedagogical C5.0 and our algorithm, and seven for DeepRED). We can see that the pedagogical baseline needs very little time to generate rules. Additionally, our algorithm needs significantly less time and memory to build rule sets than DeepRED (even when three runs of DeepRED are missing here because they exceeded the time limit of one week). We see that DeepRED is very time and memory expensive.

![Figure 16: Comparing time and memory of the three algorithms for model DNN1 on a logarithmic scale.](image)
Fig. 17 shows the accuracy and fidelity of the three algorithms on the task of classifying for ER expression with two classes. Compared to both baselines, our algorithm on average generates more accurate rules, which represent the behaviour of the neural network better (see fidelity). We want to emphasise the fact that our algorithm outperforms the pedagogical baseline in both of these measures. Rule sets that are extracted with DeepRED are the least accurate and have the lowest level of fidelity.

Fig. 17: Comparing the accuracy and fidelity of rules that are extracted with the three algorithms for model $DNN_1$.

Fig. 18 expresses the comprehensibility of the generated rule sets with the rule set size and the average rule length. Due to the significant variance across the number of rules, the horizontal axis is displayed on a logarithmic scale. We can see an outlier for our algorithm with more than 280,000 rules. DeepRED also has, though not so obvious, one outlier with more than 80,000 rules. There is no clear explanation for why this happens. Both of these data points represent rule sets that were generated on the same fold, so this would suggest the size of the set is correlated with the type of data, but further studies are required to confirm this. If we look at other values, as expected, the pedagogical baseline provides the smallest rule sets with the shortest rules. On average, DeepRED builds the longest rules.

To illustrate the correlation between the comprehensibility measure and the accuracy/fidelity, we first compare the rule set size and the average rule length with the rule set accuracy on Figs. 19 and 20, respectively. Due to the significant variance across the number of rules, this feature is displayed on a logarithmic scale. Both figures express similar trends, which confirm the following statements:

- pedagogical baseline extracts the smallest rule sets with the shortest rules,
- on average, our algorithm extracts rules with the highest accuracy, and
- DeepRED extracts the least accurate rules.
Figure 18: Comparing rule set size and average rule length of rules that are extracted with the three algorithms for model $DNN_1$.

Figure 19: Comparing rule set size and accuracy of rules that are extracted with the three algorithms for model $DNN_1$ (horizontal axis is displayed on a logarithmic scale).
Figure 20: Comparing average rule length and accuracy of rules that are extracted with the three algorithms for model DNN.1.

Next, we compare the rule set size and the average rule length with the rule set fidelity on Figs. 21 and 22, respectively. Due to the significant variance across the number of rules, this feature is displayed on a logarithmic scale. Both figures express similar trends, which confirm the following statements:

- pedagogical baseline extracts the smallest rule sets with the shortest rules,
- our algorithm extracts rules with the highest level of fidelity, and
- on average, DeepRED extracts rules with the lowest level of fidelity.

Our results confirm that decompositional algorithms generate longer rules, but it is not always true that these rules mimic the behaviour of the neural network better. In our case, DeepRED (i.e., decompositional baseline) extracts rule sets that have lower fidelity than those extracted with the pedagogical baseline. Our algorithm, on the other hand, outperforms the pedagogical baseline in terms of fidelity and average accuracy.
Figure 21: Comparing rule set size and fidelity of rules that are extracted with the three algorithms for model $DNN_1$ (horizontal axis is displayed on a logarithmic scale).

Figure 22: Comparing average rule length and fidelity of rules that are extracted with the three algorithms for model $DNN_1$. 

55
All measures that we used so far for the evaluation are quantitative. For quality, we want to know how similar the rules extracted with different algorithms are in terms of features. The three algorithms return sets of rules, which have conditions on the input features and explain the corresponding networks. Therefore, we compare features that appear in the rule sets, built with different algorithms, to find the ones that are crucial in explaining the network.

Because the pedagogical baseline generates significantly smaller rule sets and consequently uses significantly fewer features, compared to our algorithm and DeepRED, we first make a comparison between the two decompositional algorithms. We compare features, which appear in the rule sets that are generated by our algorithm and DeepRED, as follows. Let us denote the rule sets, generated with our algorithm as $OA_f$, where $f = 1, \ldots, 5$ (i.e., one rule set for each fold). Similarly, we denote the rule sets, generated with DeepRED as $D_f$, where $f = 1, \ldots, 5$. For each of the five folds, we extract features that appear in both rule sets, $OA_f$ and $D_f$. There are 72 distinct features across all folds. For each of these features, we calculate its frequency. A feature can appear in one, two, three, four or five folds (e.g., in this context, appearing in one fold means that the feature appeared in both rule sets, $OA_f$ and $D_f$, for some $f$). Fig. 23 illustrates the number of features that appear in different number of folds. There are three features that appear in the rule sets, generated by both algorithms, in all five folds: WWP1, NME3, and RARA.

![Figure 23: Number of features that appear in both rule sets, generated with our algorithm and DeepRED, in all five folds or in a subset of folds (for model DNN$_1$).](image)

From this analysis, we got a sense of which features are important. Next, we include the pedagogical C5.0 algorithm into the same analysis. We denote the rule sets, generated with the pedagogical C5.0 algorithm as $P_f$, where $f = 1, \ldots, 5$. For each of the five folds $f$, we look at the intersections of features in rule sets $OA_f$, $D_f$, and $P_f$. A feature can appear in one, two, three, four or five folds. In this context, appearing in one fold means that the feature appeared in all three rule sets, that is, $OA_f$, $D_f$, and $P_f$, for some $f$. Fig. 24 shows these features with a Venn diagram. One feature appears in all folds, that is, WWP1. We can conclude that this is the most important feature in predicting
the ER expression, but to confirm, an opinion of a specialist is needed, which can be
done in further studies. Feature NME3 appears in three folds (i.e., 3, 4, and 5) and fea-
ture RARA appears in two folds (i.e., 4 and 5). Six features appear in all three rule sets
in one out of five folds.

Figure 24: Venn diagram, representing features that appear in all three rule sets (gen-
erated by our algorithm and the two baselines) across five folds (for model $DNN_1$).

In this chapter, we discussed the worst-case theoretical complexity of our algo-
rithm, which is exponential. In the analysis, we ignored the algorithm’s optimisation of
using only important features to build rules, because there is no estimate of how many
features are cut in the process. Next, we compared the performance of our algorithm
with two baselines in Section 5.2.

Results demonstrate that our decompositional algorithm better represents the be-
haviour of the neural network from which it extracts rules than the pedagogical base-
line. Algorithms of the pedagogical approach have the advantage of being more time
and memory efficient, which our results confirm. Nevertheless, if the goal is to mimic
the network in the best way and resources are available, it is better to use a decompo-
sitional algorithm.

Furthermore, we compared our rule extraction algorithm to the decompositional
DeepRED, which also extracts rules from deep neural networks. Due to the DeepRED’s
bad time-wise performance on the first model $DNN_1$, we did not evaluate it on the
other two models (defined in Section 3.4). The first model was the simplest in terms of
the number of inputs and outputs of the neural network. DeepRED generated seven
out of ten rule sets, where the time limit for each was one week. We can only assume
that the rule sets that DeepRED was not able to generate in that time limit would be
larger than the ones it did. However, our algorithm produced shorter rules that were
more accurate and had higher fidelity.

The majority of the measures that we used to evaluate our algorithm were quanti-
tative. For evaluating the quality of the extracted rules, we looked at the feature inter-
section across the three algorithms for each fold. Next, we compared these features,
and concluded that feature WWP1 is the most important in predicting ER expressions.
6 Conclusion

In this section, we summarise the thesis and give suggestions for further research.

6.1 Summary

The core of this thesis introduces a scalable algorithm of the decompositional nature for extracting rules from neural networks. Our algorithm is able to extract rules from deep neural networks with multiple neurons in the output layer, which means that it is able to deal with the multi-class classification problems. In the past, the majority of algorithms for rule extraction were tested on the binary classification problems. Our algorithm combines two classification models: neural networks and decision IF-THEN rules. Its goal is to explain the decisions of a neural network with a simple set of understandable rules. This is especially useful in safety critical applications (i.e., healthcare), where it is crucial to understand the reasons that lead to a particular decision.

The algorithm consists of two parts. In both, it processes every hidden layer of a neural network in reverse order, that is, from the output layer towards the input layer. First, it extracts intermediate rules that connect two consecutive layers of a neural network. These rules describe a particular layer in terms of the neurons in the preceding layer. In the second part, the algorithm merges these rule sets and outputs a set of rules that describe the output layer in terms of the input features.

The algorithm was applied to a medical data set of breast cancer patients. We evaluated it on two different tasks of binary and multi-class classification problems of predicting ER expression (with two classes) and PAM50 feature (with five classes). We compared its performance to two baselines: pedagogical C5.0 and DeepRED. The first baseline was of the pedagogical nature, which our algorithm outperformed on the binary classification tasks in terms of how good the extracted rules mimicked the neural network's behaviour. Our algorithm outperformed DeepRED (i.e., decompositional baseline) in all aspects. The most outstanding was better time-wise performance of our algorithm.

We started off asking ourselves the following questions, which we tried to answer with the implementation of this algorithm:

- The aim of interpretability is to describe a model in a way that is understandable to humans [16]. Can we make a deep neural network more interpretable?

- A model is called explainable if it is able to summarise the reasons for its behaviour, gain the user's trust, or produce insights about the causes of its decisions [16]. How can we explain the predictions of deep neural networks?

- Can we explain deep neural networks’ decisions by taking into account all their neurons in an efficient way?

Our algorithm is applicable to deep neural networks and extracts a set of rules, which explain the predictions of a corresponding neural network. The extracted rules are simple, easily understandable IF-THEN rules, which clearly state what input features their prediction is made on. Overall, the experimental results show that the algorithm can efficiently extract rules, which approximate the behaviour of neural networks well.
6.2 Future work

Our algorithm outperformed a state-of-the-art algorithm of the decompositional nature, DeepRED, which is very promising and shows a lot of potential. In this section, we discuss a few ideas for further research that would improve our algorithm.

First, the algorithm's prediction for a new example is determined as a majority vote of all applicable rules. One potential improvement would be to weigh rules with confidence values, indicating their reliability. The confidence value of a rule \( r \) is calculated as:

\[
\text{confidence value}(r) = \frac{n_{\text{corr}} + 1}{n_{\text{cov}} + 2}
\]  

(6.1)

where \( n_{\text{corr}} \) represents the number of training examples that are correctly classified by the rule, and \( n_{\text{cov}} \) represents a total number of training examples that are covered by the rule. In this context, reliability represents the proportion of correctly classified examples that are covered by the rule. This way, we would prefer rules that are more reliable, which could lead to better predictions in terms of accuracy and fidelity.

A further interesting analysis would be to substitute the classification algorithm C5.0, which our algorithm uses for building rules that connect two consecutive layers, with another classification algorithm. It would be interesting to see if creating a set of rules directly, instead of first building decision trees and then extracting rules, would improve the time complexity of the algorithm. One such example is the CN2 induction algorithm, presented in [10].

Finally, we think it would be worth exploring why there are such large differences in the number of rules across different folds in the same neural network. This may be data dependent (activation values of a trained neural network), but further studies should be done to confirm or refute this. Another solution for a large number of rules is pruning the rules after extracting. Again, this is left for future work.
7 Razširjeni povzetek v slovenščini

7.1 Motivacija


Tako pedagoški kot dekompozicijski pristop imata nekaj prednosti in slabosti. Glede računske zahtevnosti so pedagoški algoritmi preprostejši, saj upoštevajo le vhodni in izhodni nivo nevronskih mrež. Slaba stran tega pristopa je izguba ogromne količine informacij iz skritih nivojev. Dekompozicijski algoritmi pogledajo vsak posamezen neuron in združijo rezultate, pridobljene iz posameznih neuronov, tako da predstavijo nevronsko mrežo kot celoto. To zahteva veliko časa in spomina, ki se z večjim številom skritih nivojev znatno poveča. Če bi lahko bolje obvladali časovno in prostorsko zahtevnost dekompozicijskih algoritmov, le-ti bolje razložijo nevronske mreže, saj upoštevajo njihovo celotno strukturo.


7.2 Cilji

Jedro tega magistrskega dela je predstavitev novega dekompozicijskega algoritma za ekstrakcijo pravil iz globokih nevronskih mrež. Izvedena pravila so oblike
s testi (pogoji) na vhodnih atributih in napovedujejo ciljno spremenljivko (razred). Tako posnemajo obnašanje nevronskih mrež in pojasnjujejo njihove odločitve. Celotno raziskavo in implementacijo algoritma so vodila sledeča raziskovalna vprašanja:

- Namen interpretabilnosti (angl. **interpretability**) je opisati model na način, ki je ljudem razumljiv [16]. Kako lahko naredimo globoke nevronске mrež bolj interpretabilne?

- Model se imenuje razložljiv (angl. **explainable**), če je sposoben povzeti razloge za svoje vedenje, pridobiti zaupanje uporabnika ali omogočiti vpogled v vzroke svojih odločitev [16]. Kako lahko razložimo odločitve globokih nevronskih mrež?

- Ali lahko razložimo odločitve globokih nevronskih mrež na učinkovit način, pri čemer upoštevamo celotno strukturo mreže?

V tem magistrskem delu predlagamo nov dekompozicijski algoritem za ekstrakcijo pravil iz globokih nevronskih mrež, ki je sposoben reševanja veččiljne (angl. **multi-class**) klasifikacije. V primerjavi z algoritmom DeepRED je naš algoritem časovno in prostorsko učinkovitejši, hkrati pa je tudi bolj razširiljiv. Glede na to, da je DeepRED edini algoritem, ki je uporaben na globokih nevronskih mrežah, z razvojem boljšega algoritma je slednji najučinkovitejši dekompozicijski pristop do sedaj, kolikor nam je znano.

Uspešnost izvlečenih pravil s predstavljenim algoritem ocenimo s sedmimi merami, hkrati pa jih primerjamo s pravili, zgrajenimi z dvema osnovnicama (angl. baselines): pedagoškim algoritem C5.0 in dekompozicijskim algoritem DeepRED. Pedagoški algoritem C5.0 definiramo kot uporabo klasifikacijskega algoritma C5.0 (opisan v [6] in [24]) na vhodnih podatkih in izhodu nevronse mreže. DeepRED je dekompozicijski algoritem, predstavljen v [52] in [53] (vir kode: [13]).

Naš algoritem je uporaben predvsem v aplikacijah, kjer je potreba po razumevanju razlogov, ki vodijo do odločitev. Dober primer je zdravstvo, kjer se lahko uporabi za razlago odločitev sistemov za podporo kliničnim odločitvam. Zdravniki se bodo na mrež veliko lažje odločili, ali se strinjajo z odločitvijo sistema ali ne, če imajo vpogled v razloge, ki so privedli do odločitve.

V naslednjem poglavju predstavimo globoke usmerjene nevronse mreže, iz katereh naš algoritem izvlecne pravila, ki aproksimirajo pripadajoče mreže.

### 7.3 Globoke usmerjene nevronse mreže

Globoke usmerjene nevronse mreže (angl. **deep feedforward neural networks**) so eden izmed osnovnih modelov globokega učenja. Sestavljene so iz nevronov, ki so razporejeni v več nivojev. Ti nivoji so povezani tako, da je izhod vsakega nevrona v prejšnjem nivoju povezan z vhodom vsakega nevrona v naslednjem nivoju. Prvi nivo se imenuje vhodni nivo, kjer vsak nevron ponazarja en vhodni atribut. Nato sledi poljubno število (več kot en) skritih nivojev in izhodni nivo. Cilj mreže je aproksimirati neko funkcijo, ki je predstavljena s podatkovnimi točkami na določenih atributih.
Namen povezave, ki poteka od nevrona \( i \) do nevrona \( j \), je prenesti aktivacijsko vrednost nevrona \( i \), t.j. \( a_i \), do nevrona \( j \). Vsaka povezava ima pripisano utež \( w_{i,j} \) in vsak nevron ima pripisano neko vrednost \( a_0 \), ki predstavlja njegovo pristranskost (angl. bias). Slika 25 predstavlja matematični model nevrona \( j \). Njegova aktivacijska vrednost \( a_j \) je izračunana na sledeč način. Če ima nevron \( n \) vhodnih povezav, se najprej izračuna utežena vsota vhodnih podatkov:

\[
in_j = \sum_{i=0}^{n} w_{i,j} a_i.
\]

(7.1)

Nato se na \( in_j \) uporabi še aktivacijska funkcija \( g \):

\[
a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i).
\]

(7.2)

Slika 25: Matematični model nevrona [37].

Namen aktivacijske funkcije \( g \) je vpeljati nelinearnost v nevronsko mrežo, kar jim zagotovi sposobnost predstavitve nelinearnih funkcij. Če je v izhodnem nivou \( c \) nevronov, nevronsko mreža za določen nabor vhodov \( x_1, x_2, \ldots, x_m \) vrne določene izhode \( y_j \in \{\lambda_1, \lambda_2, \ldots, \lambda_c\} \), ki se lahko razlikujejo od pravih izhodov. Za napoved pravilnih izhodov, nevronska mreža v procesu učenja spreminja uteži povezav s t.i. vzvratnim razširjanjem (angl. back-propagation) (povzeto po [37]).

### 7.4 Načrt za evaluacijo algoritma

skupine glede na estrogenski receptor (pozitiven ali negativen) in PAM50 test (pet pod-
tipov). Obe skupini uporabimo kot ciljni spremenljivki. Uporabljamo 5-kratno prečno
preverjanje, kjer so podatki razdeljeni glede na ciljno spremenljivko. Ključni koraki za
evaluacijo algoritma so sledeči.

(A) Po preučevanju nabora podatkov METABRIC se odločimo za izvedbo dveh raz-
ličnih vrst klasifikacije. Najprej rešujemo problem binarne klasifikacije, nato pa
še problem večciljne klasifikacije s petimi razredi. Izvedemo tri eksperimente:

- binarno klasifikacijo s 1.000 vhodnimi atributi,
- binarno klasifikacijo z 2.000 vhodnimi atributi in
- večciljno klasifikacijo z 2.000 vhodnimi atributi in petimi razredi.

Pri problemu binarne klasifikacije napovedujemo estrogenski receptor (ER), za
reševanje problema večciljne klasifikacije pa uporabimo atribut PAM50. Modele
za vsakega izmed teh problemov označimo kot $DNN_1$, $DNN_2$ in $DNN_3$.

(B) Izberemo arhitekturo nevronskih mrež, ki jih bomo uporabili za reševanje de-
finiranih problemov. Le-ti deloma določajo strukturo pripradajočih nevronskih
mrež (na primer: vhodni in izhodni nivo mrež sta že določena), kot je to opisano
na Sliki 26.

Za lažjo primerjavo smo se omejili na globoke nevronske mreže z dvema skri-
tima nivojema. Z metodo “grid-search” poiščemo najboljše parametre za število
nevronov v obeh skritih nivojih, število učnih iteracij (angl. epoch) in velikost
paketov (angl. batch size), ni določa koliko učnih primerov gre skozi mrežo na-
enkrat. Pri tem uporabljamo 5-kratno prečno preverjanje in izberemo parame-
tre z najboljšo povprečno napovedno točnostjo. Najboljši parametri so skupaj
z vhodnimi podatki in ciljno spremenljivko (“Izhod”) zbrane v Tabeli 14. Struk-
tura nam pove, koliko nevronov je v posameznem nivoju nevronske mreže. Na
primer: nevronska mreža za model $DNN_1$ ima strukturo 1000-128-32-2, kar po-
meni, da ima 1.000 nevronov v vhodnem nivoju, 128 nevronov v prvem skriter
nivoju, 32 nevronov v drugem skri tem nivoju in dva nevrona v izhodnem nivoju.
Zadnji stropec v tabeli predstavlja napovedno točnost na testni množici.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DNN_1$</td>
<td>mRNA</td>
<td>ER</td>
<td>1000-128-32-2</td>
<td>50</td>
<td>64</td>
<td>96,31%</td>
</tr>
<tr>
<td>$DNN_2$</td>
<td>mRNA, CNA</td>
<td>ER</td>
<td>2000-256-16-2</td>
<td>50</td>
<td>32</td>
<td>95,91%</td>
</tr>
<tr>
<td>$DNN_3$</td>
<td>mRNA, CNA</td>
<td>PAM</td>
<td>2000-256-32-5</td>
<td>100</td>
<td>64</td>
<td>75,99%</td>
</tr>
</tbody>
</table>

Tabela 14: Izbrani najboljši parametri za modele $DNN_1$, $DNN_2$ in $DNN_3$.

(C) Nevronske mreže naučimo na učni množici in napovemo ciljno spremenljivko
na testni množici.

(D) Shranimo aktivacijske vrednosti nevronskih mrež skupaj z njihovimi napovedami
na testni množici.
Slika 26: Arhitektura nevronskih mrež za tri probleme. $DNN_1$ ima 1.000 nevronov v vhodnem nivoju in dva nevrona v izhodnem nivoju; $DNN_2$ ima 2.000 nevronov v vhodnem nivoju in dva nevrona v izhodnem nivoju; $DNN_3$ ima 2.000 nevronov v vhodnem nivoju in pet nevronov v izhodnem nivoju.

(E) Podamo aktivacijske vrednosti algoritmu za ekstrakcijo pravil, ki vrne množico če-potem pravil, ki opišejo izhodni nivo z vhodnimi atributi.

(F) Izvlečena pravila posnemajo vedenje nevronsko mreže, katere aktivacijske vrednosti so bile uporabljene za njihovo ekstrakcijo. Ta pravila le aproksimirajo nevronsko mrežo in lahko napovejo drugo vrednost ciljne spremenljivke kot nevronsko mreža. Uporabimo različne mere za ocenitev teh pravil.

Za oceno izvlečene množice $R$, ki sestoji iz če-potem pravil, uporabimo sedem mer:

• napovedno točnost (angl. *accuracy*) - delež pravilno razvrščenih primerov,
• zvestobo (angl. *fidelity*) - delež primerov, katerih napovedani razred se ujema z napovedjo nevronsko mreže,
• število pravil,
• povprečno dolžino pravil,
• čas in spomin, ki sta potrebna za ekstrakcijo pravil in
• atribute v izvečenih pravilih (kateri atribute se pojavijo v različnih množicah).

Ocena množice pravil $R$ se izračuna na testnih podatkih $x_1, x_2, \ldots, x_t$, kjer vsakemu primeru $x_j$ pripada nek pravilen razred $y_j$. Za vsak primer $x_j$ pogledamo tri vrednosti:

- pravi razred: $y_j$,
- razred, ki ga je napovedala nevronsko mreža: $d_j$ in
- napoved množice pravil $R$ po principu glasovanja (angl. *majority vote*): $c_j$.

Število pravilno napovedanih primerov (kadar je $c_j$ enak $y_j$) s pravili $R$ označimo z $N_{rs,true}$. Podobno označimo z $N_{rs,nn}$ število primerov, kjer je napoved pravil $R$ enaka napovedi nevronsko mrežo (kadar je $c_j$ enak $d_j$). Potem sta napovedna točnost in zvestoba množice pravil $R$ definirani kot:

\[
\text{napovedna točnost}(R) = \frac{N_{rs,true}}{t}
\]  

\[
\text{zvestoba}(R) = \frac{N_{rs,nn}}{t}
\]

Z drugimi besedami, zvestoba nam pove, kako dobro izvečena pravila aproksimirajo nevronsko mrežo, na kateri so bila naučena.

V naslednjem poglavju predstavimo naš algoritem za ekstrakcijo pravil.

### 7.5 Algoritem za ekstrakcijo pravil

Predstavljeni algoritem se je zmožen naučiti pravil iz globokih nevronskih mrež, ki slikajo vhodne atribute v izhodni nivo mreže. Algoritem uporablja dekompozičijski pristop, kar pomeni, da poleg vhodnega in izhodnega nivoja vzamemo v obzir tudi notranjo strukturo mreže (skrite nivoje). Osnovna ideja algoritma je uporabiti aktivacijske vrednosti vseh nevronov, zato je prvi korak naučiti pripadajoče nevronsko mrežo.

Brez izgube za splošnost predpostavimo, da rešujemo problem večciljne klasifikacije s $c$ razredi $\{\lambda_1, \lambda_2, \ldots, \lambda_c\}$ in $m$ učnimi primeri $x_1, x_2, \ldots, x_m$, kjer ima vsak $x_j$ pripadajoč pravilni razred $y_j \in \{\lambda_1, \lambda_2, \ldots, \lambda_c\}$. Izhodni nivo nevronskes mreže je predstavljen kot $o = o_1, ..., o_c$ s toliko nevroni, kot je možnih razredov. Vhodni nivo z $n$ atributi je predstavljen kot $i = i_1, \ldots, i_n$. Skriti nivoji so označeni s $h_i \in \{h_1, h_2, \ldots, h_k\}$. Za vhodni in izhodni nivo velja, da je $i = h_0$ in $o = h_{k+1}$. Za nek primer $x_j$, označimo vrednost v $i$-tem skritem nivoju kot $h_i(x_j)$.

Spomnimo se oblike pravil, ki jih algoritem generira in definirajmo nov pojem:

- Pravilo dolžine $L$ z $j$ pogoji je oblike: ČE test$_1$ IN test$_2$ IN ... IN test$_{L}$ POTEM X.
- Pogoji pravila se imenujejo *izrazi* (angl. *terms*) in zavzamejo eno izmed oblik:
  - nevron $\leq$ prag ali
  - nevron $> $ prag.

7.5.1 Gradnja pravil

Eden izmed glavnih delov algoritma je gradnja odločitvenih dreves na aktivacijskih vrednostih dveh zaporednih nivojev nevronske mreže, ki jih nato pretvori v odločitvene če-potem pravila, za kar uporablja klasiﬁkacijski algoritam C5.0. V prvem delu algoritma izrazi nivo $h_i$ s pogoji na atributih v nivoju $h_{i-1}$. Kot bomo pozneje opisali, se bodo znotraj ekstrakcije pravil pojavili izrazi s pogoji na nevronih v nivoju $h_i$ (na primer: $h_i, n \leq 0, 3$). V tem primeru se ta izraz (pogoj) uporabi na aktivacijskih vrednostih v nivoju $h_i$ za vse učne primere (na primer: preveri, če je $h_i, n$ manjši ali enak pragu 0,3), kar nam da ciljno spremenljivko z vrednostmi 0 ali 1 (kar pomeni, da je bil pogoj zadoščen ali ne). Algoritem za gradnjo pravil, ki bodo slikala iz nivoja $h_{i-1}$ v nivo $h_i$, v tem primeru poda algoritmu C5.0 sledene podatke:

- aktivacijske vrednosti nivoja $h_{i-1}$ in
- izračunano ciljno spremenljivko na aktivacijskih vrednostih nivoja $h_i$, kot je opisano zgoraj.

C5.0 najprej zgradi odločitveno drevo, ki ga zatem pretvori v neurejeno množico če-potem pravil. Poleg tega algoritem izkoristi še dodatno možnost algoritma C5.0 (t.j. winnow), da le-ta uporabi samo pomembne atribute pri gradnji odločitvenih dreves.

Algoritem v prvem koraku zgradi množico pravil $R_{h_k \rightarrow o}$, ki preslikajo zadnji skriti nivo v izhodni nivo. Samo v tem koraku algoritem poda ciljno spremenljivko algoritmu C5.0 na drugačen način, kot pozneje skozi celoten potek (in kot je opisano v prejšnjem odstavku). Namesto tega mu poda ciljne vrednosti $\{\lambda_1, \lambda_2, \ldots, \lambda_c\}$. Za vsak učni primer $x_j$ algoritem pogleda vrednosti izhodnih nevronov, t.j. $h_{k+1,1}(x_j), \ldots, h_{k+1,c}(x_j)$. Če je izmed teh vrednosti največja vrednost $h_{k+1,u}(x_j)$, potem se primeru $x_j$ dodeli vrednost ciljne spremenljivke $\lambda_u$. Rezultat tega koraka je množica pravil $R_{h_k \rightarrow o}$, ki opisujejo izhodni nivo z nevroni v zadnjem skritem nivoju.

Algoritem nato pridobi unijo vseh izrazov v pravilih iz množice $R_{h_k \rightarrow o}$ in za vsakega izmed njih zgradi nova pravila, ki slikajo aktivacijske vrednosti iz prejšnjega nivoja $h_{k-1}$ v vrednosti 0 ali 1 (kot opisano na začetku tega poglavja). Vhodni podatki so aktivacijske vrednosti na nivoju $h_{k-1}$ za vse učne primere, ciljna vrednost pa se izračuna tako, da se trenuten izraz (pogoj) uporabi na aktivacijskih vrednostih na nivoju $h_k$ za vse učne primere $x_j$. Če je pogoj izpolnjen, je ciljna vrednost enaka 1, sicer pa 0. Unija vseh pravil, ki jih algoritem pridobi skozi ta postopek za vse unikatne izraze v množici pravil $R_{h_k \rightarrow o}$, je množica pravil $R_{h_{k-1} \rightarrow h_k}$.
Ta postopek se nato ponovi za vsak skriti nivo, dokler se ne zgradi množica \( R_{i \rightarrow h_1} \), ki izrazi prvi skriti nivo z vhodnimi atributi. S tem se zaključi prvi del algoritma, v katerem je bilo zgrajenih \( k + 1 \) množic pravil, ki opišejo vsak nivo nevronske mreže (razen vhodni nivo) s pogoji na nevronsih v prejšnjem nivoju:

\[
R_{i \rightarrow h_1}, R_{h_1 \rightarrow h_2}, \ldots, R_{h_{k-1} \rightarrow h_k}, R_{h_k \rightarrow o}.
\]

Cilj je pridobiti množico \( R_{i \rightarrow o} \), ki jo algoritem zgradi z združevanjem teh \( k + 1 \) vmesnih množic pravil.

### 7.5.2 Združevanje pravil

Podobno kot gradnja pravil, tudi združevanje pravil poteka od izhodnega proti vhodnemu nivoju. Algoritem najprej združi množici \( R_{h_{k-1} \rightarrow h_k} \) in \( R_{h_k \rightarrow o} \) v množico pravil \( R_{h_{k-1} \rightarrow o} \), ki slika predzadnji skriti nivo v izhodnega. V naslednjem koraku sta združeni množici \( R_{h_{k-2} \rightarrow h_{k-1}} \) in \( R_{h_{k-1} \rightarrow o} \). Rezultat tega koraka je množica \( R_{i \rightarrow o} \), ki izrazi izhodni nivo z vhodnimi atributi. V vsaki naslednji iteraciji je izhodni nivo izražen s pogoji nevronov, ki so en nivo bližje vhodnemu, dokler ni zgrajena zadnja množica \( R_{i \rightarrow o} \), ki izrazi izhodni nivo z vhodnimi atributi.

V splošnem algoritem združi množici pravil \( R_{h_{j-1} \rightarrow h_j} \) in \( R_{h_j \rightarrow o} \) tako, da vsak izraz v množici \( R_{h_{j-1} \rightarrow o} \) zamenja z vsemi pripadajočimi pravili iz množice \( R_{h_{j-1} \rightarrow h_j} \). Pri tem se upoštevajo vse možne kombinacije. Za lažje razumevanje je v poglavju 3.2 algoritem predstavljen na enostavnem primeru.

Ker pri združevanju algoritem enostavno zamenja izraze s pripadajočimi pravili, se lahko zgodi, da po združitvenem koraku dobimo pravila, ki vsebujejo odvečne (angl. redundant) izraze in logično nezadovoljiva (logično nekonsistenta, angl. unsatisfiable) pravila. Za znižanje računske kompleksnosti algoritem take izraze in pravila odstrani po vsaki združitvi dveh množic. V končni množici \( R_{i \rightarrow o} \) se tako lahko vsak vhodni atribut pojavi največ dvakrat (enkrat s simbolom ≤ in enkrat s simbolom >, pod pogojem, da se izraza ne izključujeja).

Rezultat algoritma za ekstrakcijo pravil je množica pravil, ki izrazijo izhodni nivo nevronske mreže s pogoji na vhodnih atributih.

### 7.6 Rezultati

Algoritem smo ocenili na treh zastavljenih problemih in primerjali rezultate definiranih ocenitvenih mer z dvema osnovnicama:

- pedagoškim algoritemom C5.0, ki slika vhodni nivo nevronske mreže v izhodni nivo in
- dekompozičijskim algoritemom DeepRED [52].

Ena izmed glavnih razlik med našim algoritemom in algoritemom DeepRED je ta, da naš algoritem v eni iteraciji pridobi pravila za vse razrede, medtem ko DeepRED ponovi celotni postopek za vsakega izmed razredov. DeepRED bi tako moral za model \( DNN_i \) izvedeti deset množic, t.j. dve množici (za razrede ER- in ER+) za vsako izmed petih nevronskeh mrež (zaradi 5-kratnega prečnega preverjanja). Že na tem modelu pa se je slabo odsredil, zato smo eksperimente za DeepRED izvedli samo na modelu \( DNN_i \), kjer...
je zgradil sedem od desetih množic pravil (s časovno omejitvijo en teden za gradnjo
pravil za eno nevronsko mrežo za en razred). Nepopolni rezultati za modela \( DNN_2 \) in \( DNN_3 \) so predstavljeni v Dodatku A.

Po pričakovanjih se je v večini mer najbolje odrezal pedagoški algoritem C5.0. Eks-
trakcija pravil s tem algoritmom je bila najhitreje izvedena, hkrati pa je zavzela najmanj
fizičnega spomina. Razmerje med potrebnimi časom in spominom za ekstrakcijo pravil
s tremi algoritmi je predstavljeno na Sliki 27. Zaradi 5-kratnega prečnega preverjanja
vsak algoritem zgradi pet množic pravil, razen algoritma DeepRED, ki jih zgradi deset,
ker izvleče pravila za vsak razred posebej. Ker treh od desetih množic ni uspel zgra-
diti, je prikazanih sedem rezultatov za DeepRED in po pet za vsakega izmed preostalih
algoritmov. Rezultati so zaradi velike razlike med algoritmi prikazani na logaritemski
skali.

Slika 27: Primerjava časovne in prostorske komponente treh algoritmov (pedagoški al-
goritem C5.0, naš algoritem in DeepRED) za model \( DNN_1 \).

Naš algoritem generira pravila z najvišjo napovedno točnostjo in največjo zvestobo
dmnevski mreži v primerjavi z obema osnovnicama. To je razvidno iz Slike 28. Vidimo
tudi, da se je v teh merah najslabše odrezal algoritem DeepRED. To sta edini meri, kjer
je naš algoritem boljši od pedagoške osnovnice. Če izvajamo ekstrakcijo pravil z na-
menom, da najbolje aproksimiramo nevronsko mrežo, in imamo na voljo dovolj časa
in spominskih virov, se splača izvleči pravila z našim dekompozicijskim algoritmom
namesto z enostavnim pedagoškim algoritmom C5.0.

Slika 29 prikazuje število generiranih pravil in povprečno dolžino pravil. Pedago-
ški algoritem C5.0 je generiral najmanj pravil, ki so bila obenem tudi zelo kratka. Na
sliki sta vidni dve anomaliji (angl. outliers), ena za naš algoritem in ena za DeepRED.
Nimamo zanesljive razlage, zakaj prihaja do tako velikih razlik v številu izvlečenih pravi-
val na eni nevronski mreži, vendar vemo, da je prišlo do anomalij na istih podatkih.

69
Slika 28: Primerjava napovedne točnosti in zvestobe nevronski mreži pravil, zgrajenih s tremi algoritmi za model $DNN_1$.

DeepRED generira največ pravil, ki so hkrati tudi najdaljša v primerjavi z našim algoritmom in pedagoškim algoritmom C5.0. Zaradi velike razlike v številu izvlečenih pravil je ta mera prikazana na logaritemski skali.

Slika 29: Primerjava števila izvlečenih pravil in povprečne dolžine pravil, zgrajenih s tremi algoritmi za model $DNN_1$. 
Dosedanje ocenitvene mere so kvantitativne narave. Namen algoritma za ekstrakcijo pravil je razložiti odločitve nevronske mreže, zato nas zanima, kateri vhodni atributi so na nek način najpomembnejši. V ta namen primerjamo attribute, ki se pojavijo v množicah pravil, ki so zgrajene s tremi opisanimi algoritmi (našim in dvema osnovnicama). Zaradi prečnega preverjanja imamo za model \textit{DNN$_1$} pet množic pravil za vsakega izmed algoritmov. Slika 30 prikazuje atribute, ki se pojavijo v vseh množicah pravil, zgrajenimi s trema algoritmi, za vsakega izmed petih delov (angl. \textit{folds}). Atribut WWP1 se pojavlja v vseh množicah, zato sklepamo, da je to najpomembnejši atribut za napoved estrogenskega receptorja (ER- ali ER+). Naslednja najpomembnejša atributa sta NME3 (ki se pojavlja v vseh množicah v treh od petih delov) in RARA (ki se pojavlja v vseh množicah v dveh od petih delov). Četrta atribut se pojavlja v vseh množicah, generiranih s trema algoritmi, vendar le enem izmed petih delov.

Slika 30: Vennov diagram, ki prikazuje najbolj pogost pravilo v petih različnih delih, ki se pojavijo v vseh množicah, ki so zgrajene z našim algoritmom in obema osnovnicama (za model \textit{DNN$_1$}).

Eksperimentalni rezultati potrjujejo, da so dekompozični algoritmi zahtevnejši (v smislu porabljene časa in spomina) in generirajo daljša pravila, vendar ni vedno res, da ta pravila aproksimirajo obnašanje nevronske mreže bolje. Rezultati so pokazali, da dekompozična osnovnica DeepRED izvleče pravila, ki so manj zvesta nevronski mreži, kot tista, ki so izvedena z enostavnim pedagoškim algoritmom C5.0. Se pa je naš dekompozični algoritem odrezal bolje kot pedagoški glede na napovedno točnost in zvestobo nevronski mreži.

7.7 Zaključek

Glavni cilj magistrskega dela je bila razlaga odločitev globokih nevronskeh mrež na učinkovit način, pri čemer se upošteva celotna struktura mreže. V ta namen smo implementirali in ocenili dekompozični algoritem za ekstrakcijo pravil, ki gleda informacije iz vseh nevronov mreže, ne le na vhodnem in izhodnem nivoju. Algoritem je
uporaben na globokih nevronskih mrežah, ki rešujejo problem binarne ali večciljne klasifikacije (t.j. poljubno število nevronov v izhodnem nivoju mreže).

Eksperimentalni rezultati so pokazali, da je implementiran algoritem boljši od trenutno najboljšega dekompozicijskega algoritma DeepRED v vseh pogledih (čas in prostor, ki ga algoritem porabi, da izvleče pravila, število pravil in njihova povprečna dolžina, napovedna točnost pravil in njihova zvestoba nevronski mreži), ki je po naših podatkih edini dekompozicijski algoritem, ki se lahko uporabi za ekstrakcijo pravil iz globokih nevronskih mrež. To naredi naš predstavljeni algoritem najučinkovitejši dekompozicijski algoritem do sedaj.
Appendix A  Additional results for our rule extraction algorithm and the pedagogical C5.0 algorithm

A.1 Mapping mRNA and CNA to ER expressions

Model $DNN_2$ describes the neural network that maps a combination of 1,000 mRNA and 1,000 CNA features to the ER expression with two possible values: ER- and ER+. The best average accuracy (95.91%) over five folds was achieved with the network's structure 2000-256-16-2, 50 epochs, and batch size 32. Due to the poor time performance of the decompositional baseline, DeepRED, on the first model $DNN_1$, we present the results for model $DNN_2$ only for our algorithm and the pedagogical C5.0 baseline.

Table 15 shows the accuracy of the neural network, whose activation values on the training set were used for rule extraction with our algorithm, along with the accuracy and fidelity of the rule sets, generated with our algorithm and pedagogical C5.0. Note that due to the high number of rules in fold 1, we were not able to calculate their accuracy and fidelity in a reasonable time (annotated with *). As expected, the vanilla neural network accuracy is best (93.69%). Rules that were extracted with our algorithm are more accurate and have a higher level of fidelity, compared to the rules that were extracted with the pedagogical C5.0 algorithm. Table 16 shows the comprehensibility through the rule set size and the average rule length. Our algorithm extracts much bigger rule sets than pedagogical C5.0. The required time (in minutes) and memory (in megabytes) are presented in Table 17. Not surprisingly, the pedagogical C5.0 algorithm is faster (less than twenty seconds per rule set) and it uses very little memory. Our algorithm extracts rules in a reasonable time, that is, under eleven minutes on average.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy [%]</th>
<th>Fidelity [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NN</td>
<td>ped C5.0</td>
</tr>
<tr>
<td>1</td>
<td>90.66</td>
<td>87.88</td>
</tr>
<tr>
<td>2</td>
<td>94.19</td>
<td>91.41</td>
</tr>
<tr>
<td>3</td>
<td>93.18</td>
<td>91.92</td>
</tr>
<tr>
<td>4</td>
<td>94.44</td>
<td>88.64</td>
</tr>
<tr>
<td>5</td>
<td>95.96</td>
<td>90.40</td>
</tr>
</tbody>
</table>

Table 15: Accuracy and fidelity for model $DNN_2$.

Overall, pedagogical C5.0 extracts smaller rule sets with shorter rules faster than our algorithm. However, rules that are extracted with our algorithm are more accurate and have a higher level of fidelity, which means that they mimic the behaviour of the neural network better. If we want to have a set of rules that explain the network better and we have enough time and memory resources on hand, our algorithm is a better choice over the pedagogical C5.0 algorithm.

Lastly, we scale up the number of output neurons in the deep neural network. Both $DNN_1$ and $DNN_2$ are dealing with the binary classification task. In the next section, we provide results for the third model $DNN_3$, which classifies for five classes.
Table 16: Comprehensibility for model $DNN_2$.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Rule set size</th>
<th>Average rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ped C5.0</td>
<td>our alg.</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>4,088,725</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>47,886</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>294</td>
</tr>
<tr>
<td>Average</td>
<td>19.2</td>
<td>827,454.6</td>
</tr>
</tbody>
</table>

Table 17: Time and memory consumption for model $DNN_2$.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Time [min]</th>
<th>Memory [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ped C5.0</td>
<td>our alg.</td>
</tr>
<tr>
<td>1</td>
<td>0.31</td>
<td>40.09</td>
</tr>
<tr>
<td>2</td>
<td>0.29</td>
<td>1.96</td>
</tr>
<tr>
<td>3</td>
<td>0.27</td>
<td>8.39</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>0.28</td>
<td>1.39</td>
</tr>
<tr>
<td>Average</td>
<td>0.29</td>
<td>10.57</td>
</tr>
</tbody>
</table>

A.2 Mapping mRNA and CNA to the PAM50 feature

$DNN_3$ describes the neural network that maps a combination of 1,000 mRNA and 1,000 CNA features to PAM subtypes with five possible values: LumA, LumB, HER2-enriched, basal-like, and normal-like. The best average accuracy (75.99%) over five folds was achieved with the network's structure 2000-256-32-2, 100 epochs, and batch size 64. Due to the poor time performance of the decompositional baseline, DeepRED, on the first model $DNN_1$, we present the results for model $DNN_3$ only for our algorithm and the pedagogical C5.0 baseline.

Table 18 shows the accuracy of the neural network, whose activation values on the training set were used for rule extraction with our algorithm, along with the accuracy and fidelity of the rule sets, generated with our algorithm and pedagogical C5.0. Note that due to the high number of rules in fold 5, we were not able to calculate their accuracy and fidelity in a reasonable time (annotated with *). The neural network's accuracy is 73.66% as this is a harder task than classifying for the two classes of ER expression. With the acquired results, pedagogical C5.0 generates more accurate rules, but our algorithm has slightly higher fidelity. Table 19 shows that the pedagogical C5.0 algorithm extracts very comprehensible rules. In the average rule set, there were twelve rules, each consisting of less than four conditions. Our algorithm extracts more than four million rules per rule set with the average of 16.11 conditions per rule, which does not seem very comprehensible, but it could be improved with further pruning of the rule set. As we reported in Section 4.1, with large rule sets there is a big proportion of rules that are never applicable, so these could be removed from the rule set. Further studies are needed on how to approach further processing of extracted rules to extract compre-
hensible, but still accurate rules with a high level of fidelity. Table 20 gives the required time (in minutes) and used memory (in megabytes). Pedagogical C5.0 does not seem very affected by more input features and outputs. Our rule extraction algorithm uses, on average, almost 55 minutes to extract a rule set. However, there is a high variance in time used across folds. The minimum time needed was about 23 minutes, while the maximum was a bit over 2.5 hours.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy [%]</th>
<th>Pedagogical C5.0</th>
<th>Our Algorithm</th>
<th>Fidelity [%]</th>
<th>Pedagogical C5.0</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>78.73</td>
<td>65.57</td>
<td>60.76</td>
<td>70.89</td>
<td>65.57</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>73.42</td>
<td>62.28</td>
<td>65.32</td>
<td>64.81</td>
<td>71.65</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>72.66</td>
<td>63.04</td>
<td>60.51</td>
<td>65.82</td>
<td>65.57</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>68.86</td>
<td>61.01</td>
<td>54.94</td>
<td>62.03</td>
<td>61.77</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>74.62</td>
<td>59.64</td>
<td>*</td>
<td>62.94</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>73.66</td>
<td>62.31</td>
<td>60.38</td>
<td>65.30</td>
<td>66.14</td>
<td></td>
</tr>
</tbody>
</table>

Table 18: Accuracy and fidelity for model DNN3.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Rule set size</th>
<th>Average rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pedagogical C5.0</td>
<td>Our Algorithm</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>1,134,997</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>3,472,408</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>850,138</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>1,084,973</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>13,549,053</td>
</tr>
<tr>
<td>Average</td>
<td>12</td>
<td>4,018,313.8</td>
</tr>
</tbody>
</table>

Table 19: Comprehensibility for model DNN3.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Time [min]</th>
<th>Memory [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pedagogical C5.0</td>
<td>Our Algorithm</td>
</tr>
<tr>
<td>1</td>
<td>0.34</td>
<td>28.78</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>41.95</td>
</tr>
<tr>
<td>3</td>
<td>0.32</td>
<td>24.90</td>
</tr>
<tr>
<td>4</td>
<td>0.32</td>
<td>22.77</td>
</tr>
<tr>
<td>5</td>
<td>0.31</td>
<td>154.82</td>
</tr>
<tr>
<td>Average</td>
<td>0.32</td>
<td>54.64</td>
</tr>
</tbody>
</table>

Table 20: Time and memory consumption for model DNN3.

Our algorithm extracts rules from deep neural networks, which have conditions on the features in the input layer and map to the output layer. The algorithm takes into account the whole structure of a neural network as it splits the network into neuron levels and aggregates the results obtained from each neuron to represent the neural network.
as a whole. With these simple and understandable IF-THEN rules that approximate the behaviour of the neural network, the algorithm makes the network more interpretable and explains the network's decision, which was the main goal of this thesis.
References


