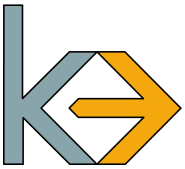
Extracting Rules from Deep Neural Networks

Master Thesis – Jan Ruben Zilke (zilke@stud.tu-darmstadt.de)

Advisor: Prof. Dr. Johannes Fürnkranz
Supervisors: Dr. Frederik Janssen & Dr. Eneldo Loza Mencía

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group



Abstract

Neural network classifiers are known to be able to learn very accurate models in a wide variety of application domains. In the recent past, researchers have even been able to train neural networks with multiple hidden layers (deep neural networks) more efficiently. However, the major downside of neural networks is that it is not trivial to understand the way how they derive their classification decisions.

To solve this problem, there has been research on extracting better understandable rules from neural networks. Although there are quite some algorithms available, to the best of our knowledge, none of them has ever been explicitly tested on deep neural networks. Furthermore, most authors focus on nets with only one single hidden layer, and hardly any paper even mentions deep neural networks.

The present thesis fills in this gap and analyses the specific challenges of extracting rules from deep neural networks. A new algorithm – DeepRED – is presented that is able to perform this task. Its main strategy is to utilize C4.5 to extract layer-wise rules that are getting merged afterwards. The evaluation of the proposed algorithm shows its ability to outperform a baseline on several tasks. This work also provides general instructions on how to use an arbitrary deep neural network rule extraction algorithm to analyse single neurons in any neural network.

Zusammenfassung

Es ist bekannt, dass neuronale Netze sehr präzise Modelle auf Basis verschiedenster Klassifikationsprobleme lernen können. In jüngster Vergangenheit gelang es Forschern sogar, auch neuronale Netze mit mehreren versteckten Schichten von Neuronen (tiefe neuronale Netze) effizienter zu trainieren. Der Nachteil neuronaler Netze bleibt aber weiterhin bestehen: Es ist nicht trivial, zu verstehen, wie deren Entscheidungsprozesse vonstatten gehen.

Um dieses Problem zu lösen, wurde an der Extraktion von (besser verständlichen) Regeln aus neuronalen Netzen geforscht. Trotz der beachtlichen Menge verfügbarer Algorithmen, ist uns kein Ansatz bekannt, der explizit auf tiefen neuronalen Netzen getestet wurde. Die meisten Autoren konzentrieren sich ausschließlich auf Netze mit nur einer versteckten Schicht von Neuronen. Insgesamt wird in kaum einer Publikation das Thema tiefe neuronale Netze überhaupt erwähnt.

Die vorliegende Arbeit schließt diese Lücke und analysiert die charakteristischen Herausforderungen der Regelextraktion aus tiefen neuronalen Netzen. Es wird ein neuer Algorithmus – DeepRED – vorgestellt, der diese Aufgabe bewältigen kann. Der Ansatz von DeepRED ist es, mit Hilfe des C4.5-Algorithmus schichtweise Regeln zu extrahieren und diese anschließend zusammenzufassen. Die Evaluation zeigt, dass die Ergebnisse von DeepRED in einigen Fällen besser sind, als die eines Vergleichsalgorithmus. Die Arbeit stellt darüber hinaus ein allgemeines Verfahren vor, wie einzelne Neuronen in einem neuronalen Netz mit Hilfe eines beliebigen Regelextraktionsalgorithmus analysiert werden können. Der verwendete Algorithmus muss dafür lediglich die prinzipielle Fähigkeit besitzen, Regeln aus tiefen neuronalen Netzen zu extrahieren.

Contents

List of Figures

List of Tables

1 Introduction

To tackle classification problems, i.e. deciding whether or not a data instance belongs to a specific class, machine learning offers a wide variety of methods. If the only goal is to accurately assign correct classes to new, unseen data, neural networks often are a good choice. They are recently becoming more famous and, in fact, for a variety of problems in different application areas authors report very low error rates that yet couldn't be achieved by other machine learning techniques [Johansson et al., 2006].

Neural networks are very powerful structures based on the human brain's functionality – sufficiently large nets, so-called deep neural networks, are able to realize arbitrary functions. And lately researchers improved the training of these structures such that they can generalize better and better from training data, for instance in the research fields of speech recognition and computer vision.

However, there is a major downside of neural networks, i.e. "humans can neither construct nor understand neural networks" [Russell and Norvig, 1995]. Since there are automated training mechanisms to build them, construction does not necessarily has to be a problem. Understanding how a neural network comes to its decision, though, can be an issue. For instance, this is the case in safety-critical application domains like medicine, power systems, or financial markets, where a hidden malfunction could lead to life-threatening actions or enormous economic loss.

Secondly, it is not possible to profit from efficient representations that might be formed by neural networks if they only can be viewed as black boxes. However, learning from the way they derive and store knowledge could be beneficial for future machine learning techniques. Making neural networks more transparent, for instance, could help to discover so-called hidden features that might be formed in deep neural networks while learning. Such features are not present in the plain input data, but emerge from combining them in a useful way.

In contrast to neural networks, rule-based approaches like decision trees or simple IF-THEN rules are known to be easily understandable. Since most humans tackle classification problems in a manner very similar to the one implemented by many rule-based techniques, their models and decision processes are more comprehensible.

To overcome the weakness of neural networks being black boxes, researchers have come up with the idea of extracting rules from neural networks. Most authors focus on extracting rules that are as comprehensible as possible while at the same time they should mimic the neural network's behaviour as accurate as possible. Especially in the 1990 much work was done in this area. Since then, a lot of rule extraction techniques have been developed and evaluated – and for many approaches quite good results have been reported.

However, most algorithms to extract rules focus on small neural networks with only one hidden layer. Surprisingly little work has been done on analysing the challenges of extracting rules from the more complex deep neural networks. Although some authors have presented rather general approaches, to the best of our knowledge, there does not exist any algorithm that has explicitly been tested on the task

of extracting rules from deep neural networks. But indeed, having a large amount of layers makes the extraction of comprehensible rules more difficult.

Therefore, the goal of this thesis is to analyse the problem of extracting rules from deep neural networks to make their decision processes more comprehensible for humans. The three main contributions of this work are the proposal of a new algorithm that is able to successfully extract understandable rules from such networks, an extensive analysis of deep neural network rule extraction using our algorithm, and general instructions on how our approach can be used to analyse single neurons in arbitrary neural networks. The thesis is structured as follows:

Chapter 2 introduces the machine learning terms and definitions that are important for the context of rule extraction from deep neural networks. This includes the classification problem in general as well as rule-based learning methods and neural networks. Afterwards, in Chapter 3, we take a look at the current state-of-the-art in extracting rules from neural networks. Here, we define the problem as well as the basic approaches to tackle it, and we present some existing rule extraction algorithms. Chapter 4, then discusses the specific challenges when working with deep neural networks. At this point, we also propose a new algorithm that can successfully extract rules from these more complex neural networks. Using several datasets and different settings, the algorithm is being evaluated in Chapter 5. Here, we also explain our experiments and discuss the results.

2 Machine Learning Fundamentals

In this chapter, we want to give an overview of the main machine learning structures and define the relevant terms. The information provided in this chapter is meant to help understanding the essential concepts the subsequent parts of this thesis are based on.

Machine learning (ML) is a research field that aims at creating automated methods of data analysis [Murphy, 2012]. The machine learning methods should be able "to adapt to new circumstances and to detect and extrapolate patterns" [Russell and Norvig, 1995]. Typically, ML is divided into three groups, i.e. *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised learning tries to learn a mapping from some input data to some outputs, while the correct mapping of the data used for learning is known [Murphy, 2012]. Therefore, the aim is to reflect the model inherent in the data. There are basically two supervised learning problems, that are *regression* and *classification*. The latter is the problem most relevant for this thesis and will be described more extensively in the subsequent section. There, we also shortly introduce the regression problem.

On the other hand, there is unsupervised learning which aims at finding interesting patterns in data, without having the possibility to check if the found patterns are correct in any sense. The third group inside machine learning is reinforcement learning, where the goal is to learn "how to act or behave when given occasional reward or punishment signals" [Murphy, 2012].

The chapter is structured as follows: In Section 2.1, we introduce the classification problem, which is the most important for this thesis. After describing the relevant terms and ideas, we take a closer look at two general approaches to solve classification problems: Section 2.2 gives an overview of rule-based methods, while in Section 2.3 neural networks are presented.

2.1 The Classification Problem

Classification describes the task of assigning a category to an object of interest. The object of interest, in literature often referred to as *instance* or *example*, e.g. might be a picture of a handwritten number like the ones depicted in Figure 2.1. So called *attributes* or *features* are used to form an instance. In the case of our handwritten number picture every single pixel would be an attribute – with the pixel's colour being the attribute's value. One could also think of other examples, where not all attributes take on values from the same domain.

In literature, typically two kinds of feature types are distinguished: *discrete* and *continuous* features. Discrete attributes are from a finite domain, while continuous features can take on any value of an infinite domain. The discrete attributes can further be distinguished into the categories *categorical* and *ordinal*. The values of the latter can be put into an order while this is not the case for categorical attributes. If only discrete attributes can be processed by an algorithm, continuous ones can get *discretized* which means that these values are converted into ordinal features.

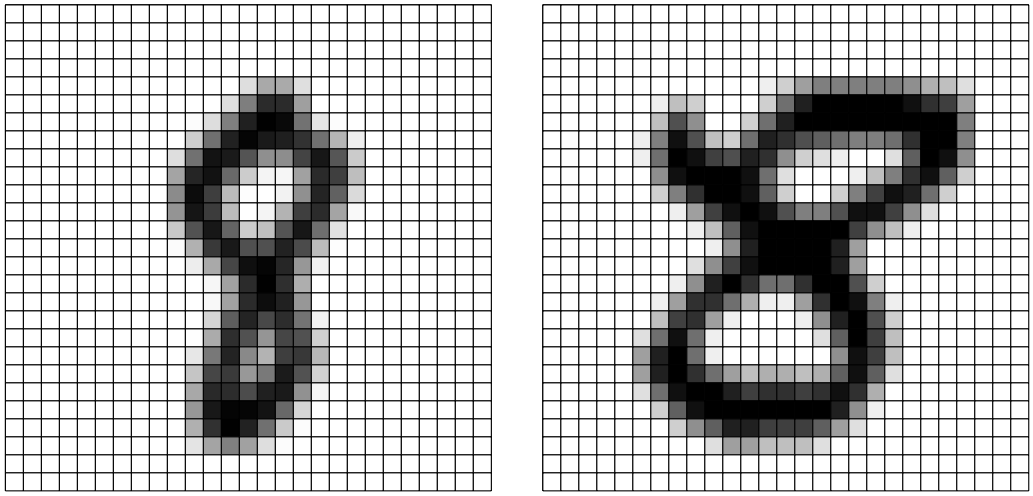


Figure 2.1: Two examples from the MNIST dataset (cf. Section 5.1.1), each consisting of 784 attributes (pixels), while both examples belong to the class *8*.

Summarised more formally[1] an instance $\boldsymbol{x}$ described by k attributes is defined as

$$\boldsymbol{x} = (x_1, ..., x_k) \in \mathbb{D} \tag{2.1}$$

while $\mathbb{D}$ is defined as $\mathbb{D} = A_1 \times ... \times A_k$. To refer to the jth attribute of an instance $\boldsymbol{x}_i$, for the remainder of this thesis, we are going to use the notation $x_{i,j}$.

As mentioned earlier, classification assigns a category to every instance $\boldsymbol{x}$. In literature, typically the terms *class* or *label* are used to describe the category. In our handwritten number example an instance's class might be one of the numbers $0, 1, 2, 3, 4, 5, 6, 7, 8$ or 9. More generally the class $\boldsymbol{y}$ of an instance $\boldsymbol{x}$ is one of the u elements in $\mathbb{L} = \{y_1, ..., y_u\}$.

Having defined the relevant terms, we are now able to formulate the classification problem as a mathematical function:

$$f : \mathbb{D} \rightarrow \mathbb{L} \tag{2.2}$$

In a nutshell: Given an input vector of attributes describing an instance $\boldsymbol{x} \in \mathbb{D}$, a so called *classifier* f returns one single class $\boldsymbol{y}$ of a finite set of discrete classes $\mathbb{L}$ [Bishop, 2006].

Besides the handwritten numbers example, one could think of other classification problems like specifying e-mails as spam or non-spam (based on the words in the message, the sender, and the recipients), labelling the current weather (given the temperature, humidity, and cloudiness), or classifying cancer (having a patient's screening parameters). Many more application areas are conceivable or have already been analysed using classification.

Before exploring different approaches to approximate a classifier function, we first need to consider some variants of the classification problem as well as some terms closely related.

[1] Please note that the definitions introduced in this chapter are based on the notation used in [Janssen, 2012].

2.1.1 Variations and Related Terms

We defined the output of a classifier to be a single class. However, it is also possible to define the output to be a subset of all possible classes, i.e. an instance could belong to more than one class. If $\mathbb{L}$ is defined this way, we call it a *multi-label classification problem* [Loza Mencía, 2012].

In our definition, we also didn't restrict the number u of possible classes, i.e. $\mathbb{L} = \{y_1, ..., y_u\}$. In some cases, however, it might be convenient to constrain it to $u = 2$. Problems constrained like this are called *binary classification problems* – in contrast to the unconstrained *multiclass classification problems*[2]. If we can only handle binary classification problems but the original data requires multiclass classification, we can utilize a *binarization technique*. The best-known strategies are *one-vs-all class binarization* and *pairwise class binarization*.

Probabilistic classification is a variant of the classification problem where it's not the aim to specify one single class an example belongs to, but rather for each class in $\mathbb{L}$ a degree of certainty the example belongs to it. Probabilistic classification therefore introduces a classifier's uncertainty to classification.

The classification problem is closely related to *regression* where, given an input vector $\boldsymbol{x}$, a numeric output value is calculated (rather than a discrete value in classification). Accordingly $\mathbb{L}$ does not have to be finite, i.e. $\mathbb{L} \subseteq \mathbb{R}$. As an example one could think of the problem of forecasting tomorrow's temperature, given today's weather data, as a regression problem.

Another related problem is *dimensionality reduction* which deals with, given a possibly large set of input variables, calculating a smaller set of output variables [Montavon, 2013]. Dimensionality reduction technically aims at finding a more efficient representation of an example. However, setting the number of expected attributes for an efficient representation to one, leads us back to the classification problem as defined in this section.

A last term we want to introduce shortly, is *clustering*. Clustering and classification both are parts of what in literature often is referred to as *pattern recognition*. Both tasks try to search for patterns in data that can help to group instances. In the case of classification, examples are assigned to predefined classes, while clustering tries to find so-called (not previously defined) *clusters* of examples that are similar (in any sense). As we will introduce in the subsequent section, classification usually is based on some training data where the correct classifications are known. For clustering tasks, however, there is no *correct* classification.

2.1.2 Approximating Classification Functions

In this section we want to give an overview of some techniques to approximate a classification function. More common terms used in place of approximation are *learning* or *training*. To learn a classifier $\hat{f}$, a *training set* D_{train} is available that provides a class $\boldsymbol{y}_i \in \mathbb{L}$ for every *training example* $\boldsymbol{x}_i \in \mathbb{D}$:

$$D_{train} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_t, \boldsymbol{y}_t)\} \subseteq \mathbb{D} \times \mathbb{L} \tag{2.3}$$

[2] Please note that multiclass classification shouldn't be confused with multi-label classification, which we have introduced earlier in this section.

The aim of training a classifier $\hat{f}$ is to approximate the function f by making use of the training set D_{train} such that $\hat{f}$ is able to correctly classify an unseen example $\boldsymbol{x}_i$, where the correct class $\boldsymbol{y}_i$ isn't known upfront. We are going to use the notation $\hat{\boldsymbol{y}}_i$ for the class predicted by $\hat{f}$ to distinguish it from the real class $\boldsymbol{y}_i$.

There is a wide variety of strategies to approximate f, likewise there are several ways to represent the classifier. Of particular interest for this thesis are rule-based systems and artificial neural networks which are introduced more extensively in Section 2.2 and Section 2.3, respectively. For the sake of giving an overview, at this point we shortly present some alternative classifier approaches.

A quite simple method is *k-nearest neighbours* (kNN). The algorithm just calculates what are the k training instances $\boldsymbol{x}_{i_1}, \ldots \boldsymbol{x}_{i_k}$ that are the closest to the example of interest. Then it takes a majority vote over the regarding $\boldsymbol{y}_i$ and predicts the most frequent class. The kNN algorithm belongs to the group of *non-parametric* classifiers which means that the number of model parameters grows with the amount of training data [Murphy, 2012]. In the case of *k-nearest neighbours*, an additional training example means an additional distance to calculate every time a new instance is being tested[3].

The opposite are *parametric* classifiers where only a fixed number of parameters is needed, independently from the amount of training data. One method using a parametric model is the *Support Vector Machine* (SVM) which can solve binary classification problems. While training, the SVM uses the training examples to build and adjust a model consisting of a fixed number of parameters[4], that is a subset of the training instances – the so-called *support vectors* [Murphy, 2012]. The support vectors in combination with the SVM function define a border that separates the instances from the first class from those of the second class. For a new example, the SVM just checks which side of the border the example belongs to and classifies it according to this.

Another famous class of parametric approaches is known as *Naive Bayes classifiers*. These methods work in a probabilistic manner, i.e. a probability for every class is being calculated and the most probable class is the one that becomes the prediction. Central for the *Bayesian* approaches is the assumption that all features are independent from each other[5]. Depending on the underlying model, the probability of each class y_i given an instance $\boldsymbol{x}$ is calculated by

$$P(y_i \mid \boldsymbol{x}) = P(y_i) \prod_{j=1}^{u} P(x_j \mid y_j) \tag{2.4}$$

while again we see, that there is only a fixed number of parameters necessary, i.e. a function $P(y_i)$ for every class and a function $P(x_j \mid y_j)$ for every attribute.

To measure the quality of a classifier, it is common to evaluate it on a *test set* D_{test}. The test set is similarly defined as D_{train} with the constraint that there cannot exist an instance that is part of both D_{train} and D_{test}. For the purpose of evaluation we let the classifier predict the class $\hat{\boldsymbol{y}}_i$ for every $\boldsymbol{x}_i$ that

[3] As a consequence, it is often the case that non-parametric methods become "computationally intractable for large datasets" [Murphy, 2012].

[4] The number of parameters depends on the SVM function chosen.

[5] Russell and Norvig, however, point out that in "practice, naive Bayes systems can work surprisingly well, even when the independence assumption is not true" [Russell and Norvig, 1995].

is part of D_{test}, then we compare the $\hat{\boldsymbol{y}}_i$ with the true classes $\boldsymbol{y}_i$. A perfect classifier would reach the goal of correctly classifying all instances, i.e. for all examples from the test set, $\hat{\boldsymbol{y}}_i = \boldsymbol{y}_i$ is true.

The main reason, why a test set is used to evaluate algorithms instead of using the training set, is that for an algorithm it is challenging to generalize from the examples seen in the training set. Therefore, a lack of generalization often results in a good classification quality using the training set to evaluate, but bad results for the test set – a problem also known as *overfitting*. With the use of a test set, we are able to grade a classifier's performance for predicting classes for new, unseen instances.

A simple measure for a classifier's quality is *accuracy* which is defined as the number of correctly classified test examples divided by the total number of test examples. Although, there are more elaborate measures and related terms [Janssen, 2012], in this thesis we limit ourselves to this short introduction and consider accuracy as the only relevant measure for us.

2.2 Rule-based Learning

This thesis is about extracting rules from deep neural networks, therefore it is necessary to introduce the concepts of rules and rule-based learning[6]. In this section, we want to give an overview of what rules are, what the important alternatives are, and how suitable rules can be learned.

2.2.1 Rules for Classification

Rules are very general structures that offer a quite easily understandable form to find the correct class for an example. They are very closely related with propositional logic [Russell and Norvig, 1995]. In this thesis, when speaking of rules, we mean what in literature frequently is called *logic rules*, *logical rules*, *propositional rules*, *crisp rules* [Duch et al., 2004], *IF-THEN rules* [Han et al., 2011], or *production rules* [Quinlan, 1993]. We adapt the general form from Duch et al. where a rule r is defined as

$$\text{IF } \boldsymbol{x} \in \mathbb{D}^{(i)} \text{ THEN } \hat{\boldsymbol{y}} = y_j \tag{2.5}$$

while $\mathbb{D}^{(i)} \subseteq \mathbb{D}$. We call the left part of the rule (the part between IF and THEN) the *condition* and the part after THEN the *consequence* or *implication*. If it is the case that an example $\boldsymbol{x}$ is part of the regarding subspace $\mathbb{D}^{(i)}$, the rule predicts the class y_j for this instance. If $\boldsymbol{x} \notin \mathbb{D}^{(i)}$, this rule does not make any statement for $\boldsymbol{x}$'s class.

A more common and more comprehensible notation of a rule is

$$\text{IF } L_{l_1}(x_{i_1}, v_{l_1}) \text{ AND } L_{l_2}(x_{i_2}, v_{l_2}) \text{ AND } \ldots \text{ THEN } \hat{\boldsymbol{y}} = y_j \tag{2.6}$$

with the functions L_l, for instance, being comparisons like $<$, $\leq$, $=$, $\geq$, or $>$. A typical rule, e.g. could look like that: IF $x_1 < 20$ AND $x_1 \geq 11$ AND $x_2 = 1$ THEN $\hat{\boldsymbol{y}} = y_1$. The consequence is predicted if and

[6] Of course it is also necessary to introduce the concepts behind deep neural networks. The regarding information can be found in Section 2.3.

only if all *terms* (the comparisons L_l) of the condition are true. Again, if at least one term is evaluated to false for the given data, no conclusion for $\boldsymbol{x}$'s class can be formed by this rule.

For some purposes, it might be useful to introduce an alternative way to describe such a rule, i.e. as a 2-tuple $r = (\{t_1, t_2, \dots\}, y_j)$ consisting of a set of terms and the consequence. For the example stated above, the equivalent tuple would be $r = (\{x_1 < 20, x_1 \geq 11, x_2 = 1\}, y_1)$.

To correctly classify examples, i.e. to approximate f as good as possible, often a single rule (per class) is not sufficient. Typically, a whole set of rules $R = \{r_1, r_2, \dots\}$ is used to build a classifier. Since with multiple rules ambiguous classifications can be the result, there are strategies to create a unique classification for each example. *Rule lists* are a simple way to achieve this goal. Here, the rules in the rule set are ranked. To classify an example, the highest-ranked rule whose condition is satisfied is being applied.

When there is no ranking of rules available, another strategy is to take the implications of all rules where the condition for an instance is satisfied and use the most frequent class as the classification. Of course, mixtures of the rule list approach and the majority vote are conceivable, as well as alternative approaches.

To be able to present a classification for every example possible, rule sets can have a *default rule*, i.e. ELSE $\hat{\boldsymbol{y}} = y_j$. This rule's consequence is predicted iff the example isn't *covered* by any other rule, i.e the condition of no other rule in the rule set is satisfied [Han et al., 2011].

2.2.2 Related and Alternative Models

There exist some classifier concepts that are very closely related to rules in the form introduced. A very popular one is the *decision tree*. This is "a structure, that is either a *leaf*, indicating a class, or a *decision node* that specifies some test to be carried out on a single attribute value, with one branch and subtree for each possible outcome of the test" [Quinlan, 1993] – the tests are also called *split points*. Figure 2.2 provides an example of a decision tree.

Decision trees can easily get converted to rule sets. For every leaf, one rule is built by simply taking each test on the path from the root to the leaf as a term for the rule's condition – and the leaf itself as the rule's implication. With this procedure, for the decision tree in Figure 2.2, we would get the following

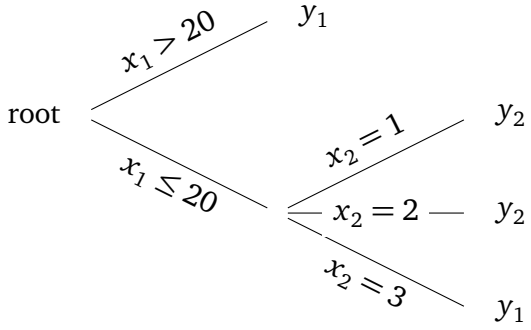


Figure 2.2: An example of a decision tree.

four rules: IF $x_1 > 20$ THEN $\hat{y} = y_1$, IF $x_1 \leq 20$ AND $x_2 = 1$ THEN $\hat{y} = y_2$, IF $x_1 \leq 20$ AND $x_2 = 2$ THEN $\hat{y} = y_2$, and IF $x_1 \leq 20$ AND $x_2 = 3$ THEN $\hat{y} = y_1$.

A second form very similar to decision trees is the concept of *decision tables*. A decision table representing the same model as depicted in Figure 2.2 can be found in Table 2.1. The condition is decoded in the upper part, in the lower part the implication/classification can be found. The left part gives an overview of the available attributes and classes, respectively [Huysmans et al., 2011]. Methods to transform decision tables to rules are also available [Vanthienen and Wets, 1994].

An extension of rules as we defined them in the preceding section, are the so-called *M-of-N rules* [Towell and Shavlik, 1993]. They can be applied when binary attributes are available or when only values close to zero or one are present, like it is often the case with connections between neurons. They realize rules that, for instance, could be written as IF M of $\{x_{i_1}, \ldots, x_{i_N}\}$ THEN $\hat{y} = y_i$, i.e. the rule's condition is satisfied if at least M of the N attributes mentioned are (close to) one. M-of-N rules can be represented by rule sets according to our definition. However, an M-of-N rule usually is a more efficient representation of such a model.

IF-THEN rules are able to model the concepts represented by decision trees, decision tables, M-of-N rules, and even concepts where these formats aren't easily applicable. The main advantage of rules and the mentioned alternative models is, that they are relatively easy to understand. For most people, a short description and a figure are sufficient to explain the representations. However, rules also have their weaknesses, e.g. some models cannot be represented efficiently by them. "Unfortunately, propositional logic is too puny a language to represent knowledge of complex environments in a concise way" [Russell and Norvig, 1995] – and so are the deeply related propositional rules.

To overcome this challenge, one could think of rules based on first-order logic or an even more expressive higher-order logic. However, in this thesis, we are going to focus on the rather simple but better comprehensible form of rules as defined in the beginning of this section. One reason is, as we will introduce later, that comprehensibility is one central goal for the extraction of rules from (deep) neural networks. Another one is the best practice in machine learning, summarized by Duch et al. as follows: "From a methodological point of view, one should always first try the simplest models based on crisp logic rules, and only if they fail, more complex forms of rules should be attempted" [Duch et al., 2004].

Nonetheless, since the algorithm presented in Section 3.2.2 uses it, at this point we shortly introduce the concept of *fuzzy rules*. Based on *fuzzy logic* they introduce "a *set membership function* that defines the degree to which an instance represents [an arbitrary] concept" [Kodratoff and Michalski, 2014], while the degree is in $[0, 1]$. This applies to the terms in the condition as well as the implication, i.e. the consequences also consist of degrees of membership to classes. Therefore, "fuzzy rules have the

Table 2.1: An example of a decision table equivalent to the decision tree depicted in Figure 2.2.

x_1	> 20	≤ 20		
x_2		1	2	3
y_1	✓			✓
y_2		✓	✓	

additional advantage over classical production rules of allowing a suitable management of vague and uncertain knowledge" [Benítez et al., 1997].

2.2.3 Learning Rules

In this section we introduce some algorithms to learn rules from training data. One can differentiate the techniques by their general approach, i.e. *divide-and-conquer* rule learning and *separate-and-conquer* rule learning [Bagallo and Haussler, 1990, Quinlan, 1993, Janssen, 2012].

Divide-and-conquer algorithms usually produce decision trees, where later rules can be derived from. When trying to find a rule for a dataset, if necessary, divide-and-conquer strategy algorithms divide the training data currently available into distinct subsets by using a split point. A split point usually is a comparison of one of the attributes available. For each subset, the algorithm again tries to find a rule – or to further divide the examples.

One of the best-known algorithms of this class is the *C4.5* algorithm [Quinlan, 1993] that proceeds as follows: If all examples in the current dataset belong to the same class, a leaf predicting this class is created. Otherwise C4.5 looks for the best split point to divide the dataset into subsets where again, C4.5 is applied to. The heuristic used to determine the best split point is the *information gain ratio* or *gain ratio*, while larger gain ratio values stand for better split points[7]. According to the split point, a decision node with the regarding subtrees is created.

The second class of approaches to create rules from a training set are the separate-and-conquer algorithms or *sequential covering* algorithms [Han et al., 2011]. Here, usually ranked rule lists are generated. Given the training data, these methods first try to find a good rule, which is added to the rule list. In the second step, the examples covered by this rule are removed from the dataset. Having this new set of instances, we repeat the process with step 1 [Han et al., 2011, Janssen, 2012]. The definition of a good rule is depending on the concrete algorithm.

RIPPER[8] is a well-known algorithm of the class of separate-and-conquer approaches [Cohen, 1995]. It extends the *IREP*[9] algorithm presented in [Fürnkranz and Widmer, 1994]. IREP adds rules to a rule list until either there are no positive examples left, i.e. instances of the class of interest, or when the best rule found is worse than a given threshold value. Central for the algorithm is, furthermore, a rule optimization method also known as *pruning*. A pseudo code of IREP is provided by Figure 2.3. The main modifications of RIPPER regarding IREP are a different heuristic to pick the best rule, another stopping criterion, and a changed rule optimization method [Cohen, 1995].

Pruning, in fact, is a central part of most modern rule extraction mechanisms. The idea is to prevent a chosen rule from overfitting the training set (cf. Section 2.1.1). A rule can be pruned by testing its performance on a *pruning set* and compare it to slightly modified versions of the rule. These rule variants can be derived by deleting terms from the rule's condition. The rule version with the best

[7] The main focus of this heuristic is to achieve a good *purity*, i.e. to form clusters where all examples belong to the same class. Since the definition of gain ratio is comprehensive and therefore would go beyond the constraints of this section, we kindly ask the reader to refer to [Han et al., 2011] if more information on this value is needed.

[8] RIPPER stands for repeated incremental pruning to produce error reduction [Cohen, 1995].

[9] IREP stands for incremental reduced error pruning [Fürnkranz and Widmer, 1994].

Input: Training set D_{train}
Output: List of rules $rules$

Split D_{train} into Pos and Neg
while $Pos \neq \emptyset$ **do**
    Split (Pos, Neg) into $(GrowPos, GrowNeg)$ and $(PrunePos, PruneNeg)$
    tmpRule = `findRule`$(GrowPos,\ GrowNeg)$
    tmpRule = `pruneRule`(tmpRule)
    if `accuracy`(tmpRule) $\leq$ *minAccuracy* **then**
        return
    else
        $Pos = Pos$—`covered`(tmpRule, Pos)
        $Neg = Neg$—`covered`(tmpRule, Neg)
        $rules[\,] =$ tmpRule
    end
end

Figure 2.3: Pseudo code of IREP (adapted from [Fürnkranz and Widmer, 1994] and [Cohen, 1995]).

performance on the (independent) pruning set is taken instead of the original rule [Han et al., 2011]. E.g. for divide-and-conquer algorithms, one can also optimize the performance by using *post-pruning* which is a similar technique to prune whole rule sets.

2.3 Artificial Neural Networks

Besides rule-based methods, *artificial neural networks* – or just *neural networks* (NN) – are the structures most important for this thesis. The function of neural networks is based on the way how the human brain works while in both cases, natural and artificial NNs, *neurons* are considered as the central elements. Although the process in the human brain is a more complex one, we will only focus on the function of the simplified artificial neural networks.

This section introduces NNs in general and how they can be used for solving classification tasks. We also discuss some related concepts that are not in the focus of this thesis, before giving a rough overview of how neural networks can be trained properly.

2.3.1 Neural Networks for Classification

A neuron is a general element that produces a single *output* by using a set of weighted *inputs* $i_1, \ldots i_k$ (with the regarding weights $w_1, \ldots w_k$), an *input function* inp, and an *activation function* act. To control the behaviour of a neuron, usually an additional input i_0 transporting the constant value of -1 is connected to the neuron. In combination with the regarding *bias weight* w_0, the threshold realized by

the function act can be modified. As the input function, often the sum of all weighted inputs is used, i.e. $\sum_j w_j i_j$.

The activation function limits the range of the neuron's output, which is often defined to be in $[0, 1]$ or $[-1, 1]$. "Roughly speaking, [a neuron] *fires* when a linear combination of its inputs exceeds some threshold" [Russell and Norvig, 1995]. This is the case because the activation usually is chosen to be a threshold-like function[10], i.e. for values less than the threshold, it produces outputs close to the minimum value, and outputs close to the maximum value for inputs that are greater than the threshold [Haykin, 1994]. Figure 2.4 shows a simple model of a neuron with k inputs. It realizes the function $act(inp(w_0 i_0, \ldots, w_k i_k))$ while the resulting value is called the *activation value* of the neuron.

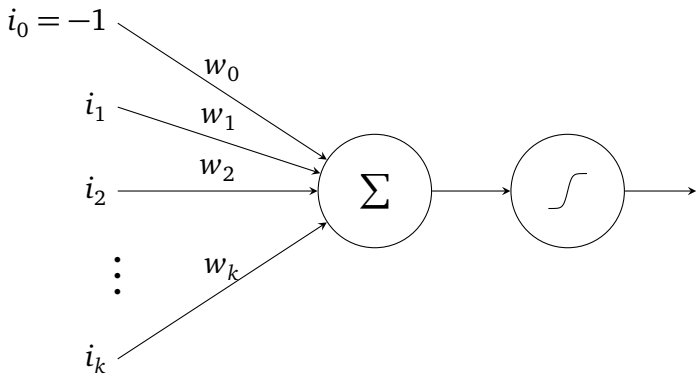


Figure 2.4: Model of a Neuron (adapted from [Haykin, 1994] and [Russell and Norvig, 1995]).

To form neural networks, a larger number of neurons as described are getting connected by directed *links* such that some neurons' outputs become the inputs of other neurons. In the context of a NN, the neurons are also called *units* or *nodes*. We limit ourselves to so-called *layered feed-forward* neural networks where units are grouped into layers. The inputs of each node are only allowed to come from the preceding layer (and therefore a neuron's output can only be linked to units in the subsequent layer) [Russell and Norvig, 1995]. For the remainder of this thesis, unless otherwise stated, we always refer to layered feed-forward NNs when mentioning neural networks.

Figure 2.5 shows an example of a feed-forward NN with four layers. The circles represent single neurons and the arrows show the links between the units. Please note that the leftmost circles (the so-called *input neurons* in the *input layer*) do not represent neurons as defined earlier. They simply provide the inputs to the neural network[11]. You might recognize, that the figure does not show any links from an input $i_0 = -1$ nor any bias weights. As shown later in this thesis (cf. Figure 5.1), a common way to depict a neuron's bias is to write the value inside the regarding circle.

[10] Famous choices for linear input functions are the *sigmoid function* $\frac{1}{1+e^{-ax}}$ and the hyperbolic tangent $1-\frac{2}{e^{2x}+1}$. If linearity is not of great importance, the threshold function or a piecewise-linear function based on the threshold function also are conceivable [Haykin, 1994].

[11] Some authors choose not to add the input layer to the total number of layers in the neural network. E.g. Russel and Norvig use another representation for the input neurons and would call the NN depicted in Figure 2.5 a three-layer neural network [Russell and Norvig, 1995]. However, we simply define the functionality of the input neurons to be different from the others.

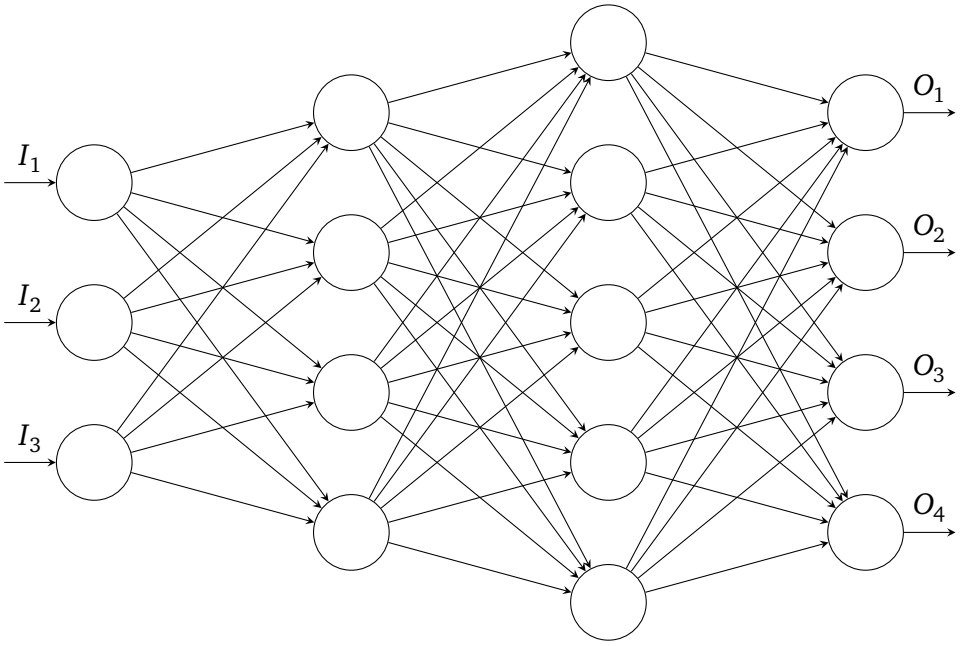


Figure 2.5: An example of a four-layer feed-forward neural network.

The rightmost layer of neurons is called the *output layer* (which contains the *output neurons*). The layers between the input and the output layer are referred to as *hidden layers*, while the regarding neurons are called *hidden neurons*. This is "because they cannot be directly observed by noting the input/output behaviour of the network" [Russell and Norvig, 1995]. If a layer appears to the left of another layer, the first one is called the *shallower* one, for instance the input layer is shallower than the output layer. On the other hand, we call the output layer *deeper* than the input layer.

The most simple neural network conceivable is the *perceptron* which only consists of a single output neuron and a set of input neurons. A *perceptron network* is a two-layer NN with an arbitrary number of output neurons. If a neural network has at least one hidden layer, it is usually referred to as a *multilayer* NN or a *multilayer perceptron* (MLP). For this thesis, we are mainly interested in handling so-called *deep neural networks* (DNN) that we define as neural networks with more than one hidden layer. For instance, the neural network depicted in Figure 2.5 is a DNN.

A neural network can be used to solve a classification problem if the inner structure and the weights are appropriately set[12]. To classify an instance $\boldsymbol{x} = (x_1, \ldots, x_k)$ we simply use the x_i as the inputs to the network. We can see, that a neural network solving a classification task with k attributes should have k input neurons.

To identify the instance's class, we need to take a closer look at the activation values of the output neurons O_i. The number of output units should be equivalent to the number of elements in $\mathbb{L}$, i.e. every output neuron represents one of the possible classes $y_1, \ldots, y_u$. The neural network classifies an example

[12] Section 2.3.3 gives some insight of how to properly adjust the weights of a neural network.

to be of class y_i iff $O_i = max_j(O_j)$, i.e. the activation value of the output neuron representing class y_i is the maximum of all output values[13].

When working with neural networks as classifiers, it is useful to know what are the limits of model representation. While a perceptron can realize any linear function, NNs with more layers are far more expressive. In fact, it is possible to show that a neural network with only one hidden layer (with a finite number of hidden neurons) can approximate any "suitably smooth function [...] to any desired level of accuracy", i.e. such a network is a *universal approximator* [Hornik, 1991, Haykin, 1994, Murphy, 2012]. Deep neural networks can even realize "any function at all" [Russell and Norvig, 1995].

Although in terms of expressive power, there is no need to introduce more than one hidden layer, in practice deep neural networks are becoming more relevant, as some recent publications in speech recognition [Jaitly et al., 2012, Seltzer et al., 2013] and object recognition [Ciresan et al., 2012, Krizhevsky et al., 2012, Szegedy et al., 2013] indicate. One reason for utilizing DNNs is that they can represent complex models more efficiently than three-layer neural networks. Even though the human brain is not a feed-forward NN, research has shown "compelling evidence that deep processing also takes place in the brain" [Russell and Norvig, 1995, Montavon, 2013].

Another one is the idea that neurons in deeper layers build so-called *hidden features*, i.e. more complex functions of the input features that can be used as a building block to solve the overall classification problem. As an example let us consider a deep neural network classifying handwritten digits (cf. Figure 2.1). A hidden neuron could fire if a circle-like structure in the upper part of the example is found, another unit might find similar shapes in the lower part. Having these hidden features, the DNN could easily distinguish eights (if both described units fire) from other digits. Furthermore, a nine could be recognized if the first hidden feature is found while no circle shape could be found in the lower part. Many more rather trivial distinctions could easily be realized by deeper layers with only having these two hidden features.

Before introducing the main strategies to train a neural network, we shortly introduce some related terms and neural network concepts that differ from our definition.

2.3.2 Related Terms and Concepts

There exist several concepts that are less restrictive in terms of the architecture of the neural network. For instance, in (*non-layered*) feed-forward NNs neurons can have inputs from any neuron and have their respective outputs linked to arbitrary neurons, as long as the neural network graph does not comprise any loops. If at least one *feedback loop* is present in a neural network, we call it *recurrent* [Haykin, 1994]. "Recurrent networks can become unstable, or oscillate, or exhibit chaotic behaviour" [Russell and Norvig, 1995]. In contrast to feed-forward neural networks, computing a recurrent NN's output is not trivial.

Hopfield networks and *Boltzmann machines* are two well-studied forms of recurrent neural networks, while both use bidirectional, symmetric weights. Each of the two NN forms have their own additional

[13] Please notice, that for multi-label classification problems the class membership has to be defined differently. However, as we limit ourselves to problems where the aim is to classify examples to exactly one class, a more general definition is not necessary.

requirements and restrictions. Since we focus on layered feed-forward neural networks, we do not go into more detail here.

A specialized kind of neural networks are the so-called *convolutional* neural networks (CNN). Using CNNs, authors recently have reported tremendous success, especially in the fields of speech recognition and object recognition [Bengio et al., 2015]. This variation of neural networks uses a *kernel* to find a single pattern in different input subsets[14]. E.g. one kernel could be used to find a circle-like structure in images, while it can be applied to different parts of the image. Due to the reuse of the kernels (with similar weights relative to the application area) CNNs need to store a much smaller number of weights, compared to standard NNs [Murphy, 2012].

2.3.3 Training Neural Networks

Training neural networks, i.e. appropriately setting a network's weights such that its prediction accuracy is maximized, is an active research field. "Learning algorithms for multilayer networks are neither efficient nor guaranteed to converge to a global optimum" [Russell and Norvig, 1995].

The *back-propagation* algorithm is the most widely used method to train a neural network [Byson and Ho, 1969]. It checks if the current setting of the neural network correctly classifies the training examples. If the classification for an example is equivalent to the true class, nothing is changed. If an instance is getting misclassified, the NN's weights need to be updated. The back-propagation algorithm proceeds in a layer-wise fashion. It first updates the weights in the output layer and continues with the next shallower level, while a degree of accountability for the errors in each output neuron is assigned to every hidden neuron. Usually, many iterations of the back-propagation algorithm are needed to appropriately adjust the weights.

Unfortunately, the approach presented works worse for more hidden layers. Therefore several strategies to support and simplify deep neural network learning are currently investigated by many researchers [Bengio et al., 2015]. One idea is to pre-train DNNs, i.e. to try to initialize the weights as good as possible – for instance, in [Jaitly et al., 2012] the authors use *Deep Belief Networks* to initialize a neural network's weights.

Another challenge is to find a good architecture for a neural network to solve a classification problem. How many layers are needed? How many neurons are necessary in each of the hidden layers? Too few layers or neurons might lead to the incapability of correctly modelling the examples. Too many neurons or layers could lead to an inefficient representation and overfitting or the overall inability to suitably train the DNN [Russell and Norvig, 1995].

Even though there are many interesting challenges regarding NN training, as well as some promising ideas to solve some problems, we limit ourselves to this brief overview of neural network training. As we will see in the next chapter, the main focus of this thesis is to handle already trained neural networks instead of training NNs by ourselves.

[14] Of course, more than one kernel can be used in a convolutional neural network.

2.4 Concluding Remarks

In this chapter, we introduced the central machine learning concepts. We defined the classification problem as well as the rule-based and the neural network approaches to handle this kind of tasks.

Section 2.1 provided the relevant definitions in the context of classification, including clarifications what instances, training sets and classifiers are. We also presented some terms and problems related to the classification task, as well as some examples of classifiers.

Rules and rule-based learning in general are important concepts for the topic of this thesis. Section 2.2 introduced IF-THEN rules, which we mainly use in the remainder of this work, as well as alternative forms like decision trees. We also gave an overview of the relevant rule learning methods.

In the last part of this chapter, we presented how neural networks can be used to solve classification problems. The definitions provided by Section 2.3 include those of neurons, (deep) neural networks, and NN layers. We also described some related terms and concepts and outlined the back-propagation algorithm to train a neural network.

This thesis analyses the task of extracting rules from deep neural networks. The subsequent chapter therefore gives an overview of the general problem and the related work available.

3 State-of-the-Art in Extracting Rules from Neural Networks

In the preceding chapter we have introduces neural networks and rule learning methods as possible approaches to tackle classification problems. In fact, both techniques are famous choices to learn models that predict the classes for new, unseen data. Recently, especially neural networks "have shown superior performance in various benchmark studies" [Huysmans et al., 2006]. For many problems, NNs can outperform rule-based learning methods if we care about accuracy.

However, neural networks have one major weakness: When it comes to comprehensibility, i.e. the ease of understanding what a trained concept is modelling, NNs are not as strong as rule-based approaches. "Concepts learned by neural networks are difficult to understand because they are represented using large assembled of real-valued parameters" [Craven and Shavlik, 1994]. This chapter introduces the idea of extracting rules from neural networks to make their decision processes better understandable.

Rule extraction is only one way to help understanding neural networks. Another famous way is to produce visualizations that give insights into a NN's behaviour. This, for instance, has been studied in [Cortez and Embrechts, 2013]. Other related work deals with the integration of knowledge (e.g. in the form of rules) into neural networks [Šíma, 1995] or the extraction of regression rules from NNs [Saito and Nakano, 2002, Setiono et al., 2002, Setiono and Thong, 2004]. Also, neural networks are not the only black box methods that researchers want to make more comprehensible. There also exist techniques to extract rules from SVMs [Diederich et al., 2010].

However, in this thesis we focus on extracting classification rules from neural networks, or more specifically DNNs. The remainder of this chapter is structured as follows: After a general introduction to rule extraction from NNs in Section 3.1, we give an overview of the current state-of-the-art and shortly introduce some interesting rule extraction approaches in Section 3.2, Section 3.3, and Section 3.4.

3.1 Introduction to Rule Extraction from Neural Networks

As shortly outlined in the beginning of this chapter, a downside of artificial neural networks is the fact, that they are not very comprehensible [Duch et al., 2004]. Their performance in classifying instances does outperform other approaches in a variety of applications [Johansson et al., 2006]. But how a neural network comes to its decision is hard to explain, or with the words of Tickle et al.: To "an end user [a neural network] is a numerical enigma comprising an arcane web of interconnected input, hidden, and output units" [Tickle et al., 1998]. In fact, the "lack of explanatory capabilities is considered as the main shortcoming of the application of neural networks" [Vellido et al., 1999]. To make them more human-comprehensible, in the 1990s a lot of research was done focussed on extracting rules from NNs, i.e. describing their behaviour or the behaviour of their components in the form of rules.

Increasing the transparency of neural networks by extracting rules has two main advantageous implications[1]. First, it gives the user some insights on how the neural network uses input variables to come to a decision – and might even expose, what in literature often is called hidden features in the NN, when rules are used to explain single neurons. Also part of the insights could be the identification of notably important attributes or the uncovering of malfunctions of the neural network. By trying to make opaque neural networks more understandable, "rule extraction techniques [bridge the] gap between accuracy and comprehensibility" and therefore resolve what in literature is well-known as the *accuracy-comprehensibility trade-off* [Craven and Shavlik, 1999, Johansson et al., 2005, Johansson et al., 2006, Sethi et al., 2012a].

Second, as a consequence of the first advantage, the better understandable form "is mandatory if, for example, the [neural network] is to be used in what are termed *safety-critical* applications such as airlines and power stations. In these cases, it is imperative that the system user should be able to validate the output of the artificial neural network under all possible input conditions" [Andrews et al., 1995]. Also, "financial regulators and law enforcement bodies require all risk management models of a financial institution to be validated" [Setiono et al., 2008]. Rule extraction techniques can be capable of validating neural network-based models and hence enlarge the application area of NNs.

To formalise the task of rule extraction from a neural network, Craven's definition summarizes the central elements: "Given a trained neural network and the data on which it was trained, produce a description of the network's hypothesis that is comprehensible yet closely approximates the network's prediction behaviour" [Craven, 1996]. A more general definition, which is not only bound to neural network rule extraction has been formulated by Huysmans et al.: "Given an opaque predictive model and the data on which it was trained, produce a description of the predictive model's hypothesis that is understandable yet closely approximates the predictive model's behavior" [Huysmans et al., 2006]. Due to the fact that in this work we limit ourselves to neural network rule extraction, here the first definition is used as our main basis.

To distinguish between the diverse approaches of extracting rules from neural networks, Andrews et al. have introduced a widely used multi-dimensional taxonomy [Andrews et al., 1995]. The first dimension they describe, is the *expressive power* of the extracted rules, i.e. their form (e.g. IF-THEN rules or fuzzy rules).

The second dimension is called *translucency* and describes the strategy an algorithm is following to extract the rules. If the technique uses a neural network only as a black box, irrespective of the NN's architecture, we call the technique *pedagogical*. If, instead, the algorithm considers the inner structure of a neural network and works on neuron-level, rather than on NN-level, we call this approach *decompositional*. If a technique uses components of both, pedagogical and decompositional approaches, the term *eclectic* is used to describe this method.

Another dimension proposed by the authors is the *quality* of the extracted rules. Since quality is a broad term, it is partitioned into multiple criteria, namely *accuracy*, *fidelity*, *consistency*, and *comprehensibility*. While accuracy measures the ability to "correctly classify previously unseen examples",

[1] In their work, Andrews et al. list additional benefits that aren't in the focus of this thesis. They mention further advantages in NN software verification and debugging, the improvement of NN generalisation, and knowledge acquisition for symbolic AI systems [Andrews et al., 1995].

fidelity measures the degree of how well the rules can "mimic the behaviour" of the neural network [Johansson et al., 2006]. Hence, fidelity can be considered as the accuracy regarding the outputs of a NN. Consistency can only be measured when the rule extraction algorithm includes a neural network training instead of processing already trained NNs: "An extracted rule set is deemed to be consistent if, under differing training sessions, the [neural network] generates rule sets which produce the same classifications of unseen examples". Comprehensibility here is only considered as a measure of the rule size, i.e. shorter and fewer rules are considered more comprehensible [Andrews et al., 1995].

At this point we want to add, that research has shown that there are better interpretable forms of rules than the plain IF-THEN format [Huysmans et al., 2011, Freitas, 2014]. Furthermore, there are eligible objections that the number of rules and their sizes is not a sophisticated enough measure of comprehensibility. In fact, Freitas has identified a number of drawbacks when using model size as the only measure of comprehensibility [Freitas, 2014]. Although there are proposals like the "prediction-explanation size" [Otero and Freitas, 2015] that might be slightly more suitable to measure comprehensibility, due to reasons of comparability and simplicity we decide to measure the understandability of rules by the simple measures mentioned first. With this decision we follow most of the former work in this area, as amongst others is mentioned in [Pazzani, 2000], [Augasta and Kathirvalavakumar, 2012b], and [Otero and Freitas, 2015].

Although Andrews et al. introduce a total number of five dimensions[2], in this thesis we are going to focus on the three criteria already presented.

In our state-of-the-art analysis we limit ourselves to algorithms having only few demands on the neural network. In accordance with [Thrun, 1993] we focus on methods that have no special requirements on how the neural network has been trained prior to rule extraction. Furthermore only algorithms are analysed that are capable of extracting rules from every feedforward neural network, despite any other architecture characteristics. As claimed by Craven and Shavlik, we want an algorithm to offer "a high level of generality" [Craven and Shavlik, 1999]. Another promising development in rule extraction research, namely the idea of making the extraction process an interactive one, has been reviewed in [Diederich et al., 2010], but is skipped in this thesis because it is beyond the scope of this work.

In the remainder of this chapter, we are going to analyse some rule extraction methods meeting the characteristics mentioned. To structure the different approaches, the translucency dimension is being used.

3.2 Decompositional Approaches

As mentioned earlier, decompositional approaches to extract rules from neural networks work on neuron-level. Usually, a decompositional method analyses each neuron and rules are formed to mimic the behaviour of this unit. The "rules extracted at the individual unit level are then aggregated to form the

[2] In fact, Tickle et al. reviewed the taxonomy and confirmed its validity using these five dimensions [Tickle et al., 1998]. However, there has been some more work on classifying methods for extracting rules from neural networks. For instance, Sethi et al. proposed a taxonomy with eight dimensions, also including the criteria introduced by Andrews et al. [Sethi et al., 2012a].

Input: Neural network NN, training examples x
Output: Set of rules $rules$

foreach *currentNeuron* $\in$ `hiddenAndOutputNeurons(`NN`)` **do**
    intermediateRules(*currentNeuron*) = `describeNeuronOutput(`*neuronInput*, x`)`
                                        `// With or without using the training examples` x
end
$rules$ = `aggregateIntermediateRules(`*intermediateRules*`)`

Figure 3.1: Pseudo code of decompositional rule extraction approaches.

composite rule base for the [neural network] as a whole" [Andrews et al., 1995]. Figure 3.1 provides pseudo code describing the general decompositional strategy of rule extraction.

Due to various reasons, we don't consider all available decompositional approaches in the subsequent overview. For instance, the algorithms described in [Craven and Shavlik, 1993], [Kane and Milgram, 1993], and [Fukumi and Akamatsu, 1998] are left out because the initial neural network fed to one of the algorithms is being retrained while or prior to the rule extraction process, which does not match with our goal of explaining a trained neural network's behaviour as it is.

3.2.1 The KT Method

One of the first decompositional approaches to extract rules from neural networks was presented in [Fu, 1994]. His KT[3] algorithm describes every neuron (layer by layer) with IF-THEN rules by heuristically searching for combinations of input attributes that exceed the neuron's threshold. A "rewriting module" is utilized to obtain rules that refer to the original input attributes rather than the outputs of the preceding layer.

To find suitable combinations the KT method applies a tree search, i.e. a rule (represented as a node in the tree) "at the ith level generates its child nodes at the $i + 1$th level by adding an additional, available attribute in all possible ways" [Fu, 1994]. Furthermore, the algorithm uses three heuristics to stop growing the tree in situations, where no further improvement is possible. The algorithm searches for both, confirming and disconfirming rules[4]. That are rules that predict a neuron to fire, if a certain input configuration is present, or not to fire, respectively. Figure 3.2 outlines the idea of the KT method to create rules that describe a neural network's behaviour.

[3] "KT is derived from the word *Knowledgetron*, which was coined by the author to refer to a neural network with knowledge" [Fu, 1994].

[4] Please note that Figure 3.2 only outlines the process of creating confirming rules. An algorithm to generate disconfirming rules can be built analogously.

Input: Neural network NN
Output: Set of rules $rules$

foreach *currentNeuron* ∈ `hiddenAndOutputNeurons(`NN`)` **do**
  setOfActivatingInputs(*currentNeuron*) =
  `findSetOfPositiveInputsExceedingThreshold(`*neuronInput*`)`
  foreach p ∈ setOfActivatingInputs(*currentNeuron*) **do**
    n = `findSetOfNegativeInputsAddedStillExceedThreshold(`p, *neuronInput*`)`
    intermediateRules[] = IF p AND NOT(n) THEN currentNeuron is activated
  end
end
$rules$ = `aggregateIntermediateRules(`*intermediateRules*`)`

Figure 3.2: Pseudo code of the KT method.

3.2.2 Transforming Neural Networks to Fuzzy Rules

In their work *Are Artificial Neural Networks Black Boxes* Benítez et al. proposed an algorithm to describe NNs by fuzzy rules [Benítez et al., 1997]. An advantage over other rule extraction approaches is that the authors invented a method which does not only approximate a neuron's behaviour, but transforms each neuron to a fuzzy rule realizing exactly the same function as the original neuron. The authors show in their work, that neural networks and "a set of fuzzy rules of a particular type" [Benítez et al., 1997] are equivalent.

The transformation of a neuron z_j results in a rule of the following form

$$r_{jk}: \text{ IF } \sum_{i=1}^{n} x_i w_{ij} + \tau_j \text{ is } A \text{ THEN } y_k = \beta_{jk} \tag{3.1}$$

where x_i are the activation values of the inputs, w_{ij} the respective weights, τ_j the neuron's threshold and A "a fuzzy set on $\mathbb{R}$ whose membership function is simply the [neuron's] activation function" [Benítez et al., 1997].

In addition to the transformation to fuzzy rules, the authors present a procedure to rewrite the rules to make them more comprehensible. To achieve this goal, they introduce a new operator, called *interactive-or* that should help interpreting a neural network's behaviour.

3.2.3 Tsukimoto's Polynomial Algorithm

Tsukimoto's approach for extracting rules from a neural network is quite similar to Fu's KT method introduced in Section 3.2.1. It also uses a layer-by-layer decompositional algorithm to extract IF-THEN rules for every single neuron, while also following the strategy of finding input configurations that exceed a neuron's threshold. The main advantage of Tsukimoto's method is its computational complexity which is polynomial, while the KT method is an exponential approach [Tsukimoto, 2000].

The algorithm achieves polynomial time complexity by searching for relevant terms using a multilinear function space. In a second step these terms are used to generate IF-THEN rules. Subsequently, if available, the training data is used to improve the rules' accuracy. In a last step, Tsukimoto's algorithm tries to optimize the comprehensibility by removing unimportant attributes from the rules. The author provides an example where, using his approach, it was possible to extract rules from a neural network representing the XOR function (cf. Section 5.1.3).

3.2.4 A Continuous/Discrete Rule Extractor via Decision Tree Induction

Another method for rule extraction was introduced in [Sato and Tsukimoto, 2001]. Using C4.5 (cf. Section 2.2.3), their algorithm CRED[5] transforms each output unit of a neural network into a decision tree where the tree's nodes are tests using the hidden layer's units and the leaves represent the class such an example would belong to (cf. Figure 3.3). Afterwards, intermediate rules are extracted from this step. In the case illustrated in Figure 3.3, if the task is to find rules for $class_1$, a single intermediate rule would be extracted, i.e. IF $h_2 > 0.5$ AND $h_1 \leq 0.6$ THEN $\hat{\boldsymbol{y}} = class_1$.

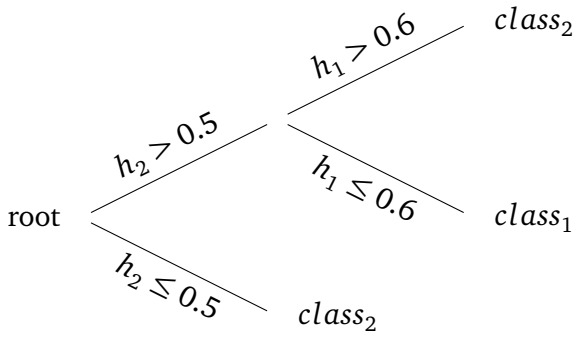


Figure 3.3: Result of CRED's first step: A decision tree providing rules for the neural network's output based on its hidden layer (adapted from [Sato and Tsukimoto, 2001]).

Next, for each split point used in these rules, another decision tree is created using split points on the neural network's input layer. In these new decision trees, the leaves don't directly decide for an example's class, but rather on the tests used in the first tree's node, as exemplary depicted in Figure 3.4. Extracting rules from this second decision tree leads us to a description of the hidden neurons' state by terms consisting of input variables. In case of the state $h_2 > 0.5$, the following intermediate rules could be extracted: IF $i_0 > 0.5$ THEN $h_2 > 0.5$, IF $i_0 \leq 0.5$ AND $i_1 = a$ THEN $h_2 > 0.5$, and IF $i_0 \leq 0.5$ AND $i_1 = b$ THEN $h_2 > 0.5$. Further decision trees must be extracted for all the other split points found in the first tree (in our example for $h_1 \leq 0.6$).

As a last step, the intermediate rules that describe the output layer by means of the hidden layer and those that describe the hidden layer based on the neural network's inputs are substituted and merged to build rules that describe the neural network's output on the basis of its inputs. The authors also implemented a rule simplification mechanism.

[5] CRED stands for continuous/discrete rule extractor via decision tree induction [Sato and Tsukimoto, 2001].

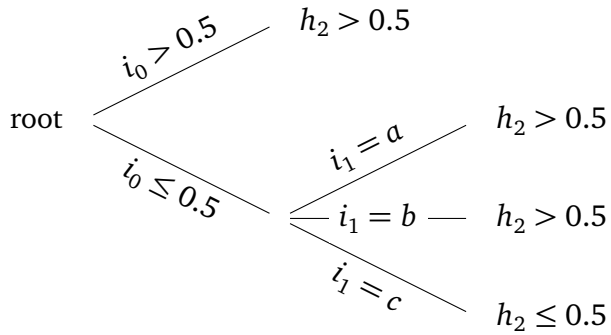


Figure 3.4: Result of one iteration of CRED's second step: A decision tree providing rules for $h_2 > 0.5$ based on the neural network's inputs (adapted from [Sato and Tsukimoto, 2001]).

Let's assume that the only rule extracted for $h_1 \leq 0.6$ is IF $i_0 \leq 0.4$ THEN $h_1 \leq 0.6$. As a result, the merging process produces the following rules to describe the neural network's behaviour: IF $i_0 \leq 0.4$ AND $i_1 = a$ THEN $\hat{y} = class_1$ and IF $i_0 \leq 0.4$ AND $i_1 = b$ THEN $\hat{y} = class_1$. Please note that unsatisfiable rules and redundant terms have been deleted from the rule set.

3.3 Pedagogical Approaches

Pedagogical approaches, in contrast to decompositional approaches, don't consider a neural network's inner structure. Or in other words, "the motif in the pedagogical approaches is to view the trained [NN] at the minimum possible level of granularity i.e. as a single entity or alternatively as a black box" [Tickle et al., 1998]. Their idea is to extract rules by directly mapping inputs to outputs [Sethi et al., 2012a].

More formally one could consider that pedagogical approaches only have access to a neural network function $NN()$. This function returns the neural network's output for an arbitrary input, but does not offer any insights into the NN's inner structure or any weights (except the number of inputs and outputs to the NN). Having $NN()$, this class of algorithms tries to find coherences between possible input variations and the outputs produced by the neural network, while some of them use the given training data and some don't.

In the preceding section we didn't present all existing decompositional approaches, and likewise we don't discuss each pedagogical approach available in more detail in the remainder of this section. E.g., the Re-RX[6] algorithm presented in [Setiono et al., 2008] is extracting rules utilizing neural networks. But instead of being the algorithm's input, the NN is only an intermediate step and therefore, in this work we don't analyse this method any further.

3.3.1 Rule Extraction based on Validity Interval Analysis

Thrun's approach makes use of *Validity Interval Analysis* (VIA), a kind of sensitivity analysis, to extract rules that mimic the behaviour of a neural network [Thrun, 1993, Thrun, 1995]. The main idea of this

[6] Re-RX stands for recursive rule extraction [Setiono et al., 2008].

method is to find input intervals where the output of the NN is stable, i.e. the predicted class is the same for slightly varying input configurations. As a result VIA produces the foundation of provably correct rules.

The algorithm picks an example (or input configuration) and randomly grows the interval of a random attribute. For each "set of validity intervals, there are two possible outcomes" of VIA, i.e. it converges or it produces a contradiction [Thrun, 1993]. If, after growing the interval, VIA still certifies its validity (if it converges), it will be saved, otherwise the interval is reset to the former setting. The process is repeated until satisfying intervals are found. The author also describes a way to form rules on the basis of validity intervals. Thrun provides several examples to illustrate his VIA-based rule extraction technique, including the extraction of rules from a neural network representing the XOR problem (cf. Section 5.1.3) and a continuous robot arm example.

3.3.2 Approaches for Rule Extraction using Sampling

This section summarizes multiple techniques, all following more or less the same strategy to extract rules from a neural network – that is creating an extensive artificial training set as a basis for rule learning. Afterwards, the sampled set of artificial data is fed to a standard rule-based learning algorithm to generate rules mimicking the neural network's behaviour. For instance, Johansson et al. investigated the advantages of what they call *oracle data* and conclude that using sampled data outperforms the usage of (only) training data in rule extraction tasks [Johansson et al., 2006]. Figure 3.5 illustrates the basic approach of this class of pedagogical rule extraction algorithms.

One of the first methods following this strategy was the Trepan algorithm [Craven and Shavlik, 1996]. Trepan proceeds quite similar to C4.5 (cf. Section 2.2.3) by searching for split points on the training data to separate instances of different classes. The main differences to C4.5 are a best-first tree expansion strategy (instead of depth-first), additional M-of-N style split points, and its ability to sample extra training examples at deeper points in the tree. As a result the algorithm also produces a decision tree which, however, could be transformed to a rule set if necessary.

Another one of these very general pedagogical approaches using sampling to extract rules from a neural network is introduced in [Taha and Ghosh, 1999]. The algorithm, called Binarized Input-Output Rule Extraction (BIO-RE), is only able to process NNs with binary or binarized input attributes. BIO-RE creates all possible input combinations and asks it to the neural network. Using the NN's output, for each example created a truth table is built. Having this extensive artificial training set, an arbitrary rule-based algorithm (cf. Section 2.2.3) can be used to extract rules that simulate the neural network's behaviour.

Input: Neural network function $NN()$, training set D_{train})
Output: Set of rules $rules$

dataSamples = `createDateset(`D_{train}`)` `// With or without using the training set` D_{train}
$rules$ = `ruleLearner(`*dataSamples*`)`

Figure 3.5: Pseudo code of rule extraction approaches using the sampling technique.

As it is possible to use any rule-producing learning algorithm, the trade-off between a correct mimicry of the original NN's behaviour and simple, well-generalized rules also depends on this choice.

ANN-DT[7] is a further sampling method on the basis of decision trees to describe a neural network's behaviour [Schmitz et al., 1999]. The general algorithm is based on CART with some variations to the original implementation. ANN-DT uses a sampling method to extend the training set such that the larger artificial training set still is representative. This is "accomplished by using a nearest neighbor method, in which the distance of a sampled point to the nearest point in the training dataset is calculated" [Schmitz et al., 1999] and compared to a reference value.

The idea of creating a large artificial instance set in a first step is also realized by the STARE[8] algorithm [Zhou et al., 2000]. Just like BIO-RE, STARE also forms extensive truth tables to train on. An advantage of STARE is its ability to not only process binary and discrete attributes, but also to work with continuous input data. To form the truth tables, the algorithm permutes the inputs, while for each continuous attribute "it is necessary to sample it across its value range with a high frequency" [Zhou et al., 2000]. In their implementation, only if necessary, the authors split a continuous input into 100 discrete inputs. In contrast to BIO-RE, STARE implements a predefined rule-extraction algorithm. The implementation focusses on finding rules using the originally discrete attributes. "Only when new rules can not be extracted out, STARE resorts to the continuous attribute that has the best clustering effect".

The last pedagogical approach using training data sampling we want to shortly introduce here is KDRuleEx[9] [Sethi et al., 2012b]. Similar to Trepan, the algorithm of Sethi et al. also generates additional training instances when the basis for the next split points is too small. KDRuleEx uses a genetic algorithm to produce these new training examples. The technique results in a decision table, which also can be transformed to e.g. IF-THEN rules, if desired.

3.3.3 Rule Extraction by Reverse Engineering the Neural Network

In their work Augasta and Kathirvalavakumar introduce a method that prunes the neural network first, before it is used for rule extraction [Augasta and Kathirvalavakumar, 2012a]. Their proposed algorithm, called RxREN[10], focusses on generating rules that are as comprehensible as possible. The NN pruning is only one indication for that. A second one is a rule pruning procedure applied at the end of the algorithm, which further simplifies the algorithm's output.

The neural network pruning is implemented by analysing the network's performance while ignoring individual input attributes. If the NN's behaviour does not alter significantly without the regarding attribute, it is pruned. The pruning step also is relevant to define the data to derive rules from, i.e. it only considers those examples that get misclassified when an input is ignored. Except for the last step, the rest of the training data has no influence on the rules.

[7] ANN-DT stands for artificial neural network decision tree algorithm [Schmitz et al., 1999].

[8] STARE stands for statistics based rule extraction [Zhou et al., 2000].

[9] We assume the name KDRuleEx is related to the fact that the algorithm extracts a total of k^d rules from a dataset having "d different features (attributes) and k different values in each feature" [Sethi et al., 2012b].

[10] RxREN stands for rule extraction by reverse engineering the neural network [Augasta and Kathirvalavakumar, 2012a].

```
Input: Neural network function NN(), training examples x
Output: Set of rules rules
foreach attribute ∈ x do
    tempX = setToZero(x, attribute)
    misclassifiedExamples(attribute) = getDifferences(NN(x), NN(tempX))
end
x = pruneInputs(x,misclassifiedExamples)
foreach currentClass ∈ classes do
    foreach attribute ∈ x do
        dataRanges(currentClass, attribute) =
        extractDataRanges(misclassifiedExamples(currentClass, attribute))
    end
    rules[] = constructRule(dataRanges(currentClass))
end
rules = pruneRules(rules)
rules = updateRules(rules, misclassifiedExamples)
```

Figure 3.6: Pseudo code of the RxREN algorithm.

In the second step, for each class and each significant attribute, RxREN computes the relevant intervals based on the misclassified examples. Such a range consists of a lower and an upper threshold that box the values that indicate a membership to the class of interest. From these intervals, rules are created. The algorithm proceeds by pruning the rules in a similar manner like the first pruning step: It analyses which rules can be deleted without relevant impact. In the last step, the authors implemented a mechanism to improve the accuracy of the remaining rules by adjusting the terms using the whole training data. A pseudo code of the RxREN algorithm is provided in Figure 3.6.

3.4 Eclectic Approaches

The third category of the translucency dimension is the group of eclectic approaches. The definition of these algorithms is less restrictive than those of the former two, which is why we don't provide a pseudo code for eclectic approaches at this point. "If rule extraction techniques incorporate elements of both pedagogical and decompositional then such techniques [are] known as eclectic techniques" [Sethi et al., 2012a]. More specifically, eclectic approaches "utilise knowledge about the internal architecture and/or weight vectors in the [neural network] to complement a symbolic learning algorithm" [Andrews et al., 1995].

The eclectic algorithms we don't consider due to different reasons include the LORE[11] algorithm [Chorowski and Zurada, 2011]. The goal of the proposed algorithm is to extract rules from a training set using NNs, instead of directly extracting rules from a neural network.

[11] LORE stands for local rule extraction [Chorowski and Zurada, 2011].

3.4.1 Extracting Refined Rules from Knowledge-based Neural Networks

The name of the MofN algorithm introduced by Towell and Shavlik is based on its ability to extract M-of-N rules (cf. Section 2.2.2) from a neural network [Towell and Shavlik, 1993]. MofN to a large amount proceeds in a decompositional fashion, trying to find rules that explain single neurons. The reason why we classify the algorithm as an eclectic approach is, that in one step it considers the significance of single attributes for the whole NN's outcome.

The main idea of MofN is to cluster the inputs of a neuron in equivalence classes, i.e. to group together inputs with more or less the same weight. After the clustering step, whole groups of inputs can get deleted if they are identified as insignificant. This could either be the case if the activation value of a whole equivalence class has no significant impact on the neuron's output or, as mentioned earlier, the NN's outcome isn't depending on these attributes. A constrained retraining of the network follows, before M-of-N rules are formed using the clusters. A final simplification step ensures better understandability of the extracted rules.

3.4.2 Fast Extraction of Rules from Neural Networks

The FERNN[12] approach proposed by Setiono and Leow first tries to identify the relevant hidden neurons as well as the relevant inputs to the network [Setiono and Leow, 2000]. For this step, a decision tree is constructed using the well-known C4.5 algorithm (cf. Section 2.2.3). The rule extraction process results in the generation of both M-of-N and IF-THEN rules.

Having a set of correctly classified training examples, FERNN analyses the activation values of each hidden unit. For each hidden unit the activation values are sorted in increasing order. The C4.5 algorithm then is used to find the best split point to form a decision tree. For the remaining datasets the same procedure is applied. All neurons that provide at least one split point for the decision tree are considered as relevant hidden units. In the next step, irrelevant neuron inputs, i.e. inputs with small weights, are removed. The decision tree forms the basis to extract the desired rules. Where possible, M-of-N rules are formed. Otherwise, IF-THEN rules are extracted[13].

3.5 Concluding Remarks

This chapter has given an overview of why we want to extract rules from neural networks and introduced the relevant measures. We also presented the state-of-the-art in NN rule extraction research.

Section 3.1 provided the motivation behind the whole topic, including a definition of rule extraction from neural networks, and gave an introduction to the widely used taxonomy to cluster rule extraction approaches. We also mentioned the criteria an algorithm should match to be included in our state-of-the-art analysis.

[12] FERNN stands for fast extraction of rules from neural networks.

[13] In fact, for continuous input variables, the authors propose the simple strategy to only extract "oblique decision rules" [Setiono and Leow, 2000] in the form of formulae like IF $(4.6x_3 + 4.2x_6 - 4.5x_7) > -1$ THEN $\hat{y} = y_i$.

The group of decompositional approaches has been presented in Section 3.2. These algorithms consider the whole neural network architecture. We introduced four available methods that match the requirements defined in Section 3.1.

The pedagogical black-box approach in general, as well as some pedagogical rule extraction algorithms have been presented in Section 3.3. We also introduced two eclectic approaches in Section 3.4.

This thesis aims at analysing the problem of extracting rules from deep neural networks. Since most of the available algorithms presented in this chapter don't mention DNNs at all, we want to take a closer look at the more specific problem in Chapter 4.

4 Extracting Rules from Deep Neural Networks

Chapter 3 has shown that there is a wide variety of algorithms to describe a one-hidden-layer neural network's behaviour by rules. However, our analysis also shows that only in a surprisingly low number of papers dealing with rule extraction, deep neural networks are even mentioned. To the best of our knowledge, there does not exist any algorithm that has explicitly been tested on the task of extracting rules from deep neural networks. Even though most pedagogical approaches should be able to perform this task without any major modifications.

In this chapter, we want to make a first step in investigating the problem of extracting rules from deep neural networks. DNNs have a few characteristics that, compared to one-hidden-layer neural networks, can complicate the rule extraction process. For instance, can more hidden layers mean a more complex extraction problem. Challenging can also be the fact, that the complexity of the concepts represented by neurons tend to be higher in deeper layers. To train a deep neural network, usually a large amount of training data needs to be available. This also has to be considered when creating rule extraction approaches.

To create such an algorithm that extracts rules from deep neural networks, we decide to build upon existing approaches. Since a comprehensive analysis would go beyond the constraints of this thesis, we can neither implement and modify every existing algorithm nor every approach presented in the preceding chapter.

To select suitable approaches, we need to find a good tradeoff between each algorithm's anticipated applicability to deep neural networks, its estimated performance (including low rule complexities and high fidelity rates) as well as its expected time complexity characteristics while also considering the effort of implementing and modifying the original algorithms to enable them to process neural networks with more than one hidden layer.

As a first step of the selection process we chose to seek for available implementations of the algorithms at the authors' webpages and publication websites. In fact, we could find a running Trepan (cf. Section 3.3.2) implementation[1] written in C code. But unfortunately we weren't able to find any other implementations of rule extraction approaches presented. Contacting the authors of the promising algorithms confirmed our assumption, that (besides Trepan) no other implementation is still available and accessible[2]. This isn't an unknown issue – already in 1999, Craven and Shavlik have criticized the problem of too low software availability [Craven and Shavlik, 1999].

Although the Trepan implementation is easy to use and there are interfaces provided to modify and extend the algorithm, the analysis of the given code results in our decision to focus on other approaches

[1] The file at http://ftp.cs.wisc.edu/machine-learning/shavlik-group/programs/trepan offers links to both, a UNIX and a Windows implementation of Trepan.

[2] At this point we would like to thank Chris Aldrich, Rudy Setiono, and Jude Shavlik for their responses and their interest in this research.

for the further research process. The main reason for this choice is the fact that an extension of the code to enable it to extract rules from deep neural networks would be similarly much effort as implementing and modifying one of the other, probably more suitable rule extraction algorithms.

Due to the fact that we cannot make use of an existing implementation that would simplify our further investigation, we decided to analyse the applicability and performance of three approaches, each from a different translucency group (cf. Chapter 3).

Section 4.1 gives some insights into our analysis process where we also present DeepRED – our proposed algorithm to extract rules from deep neural networks. The subsequent section reports on our experiences with the challenging characteristics of deep neural networks – and how DeepRED tries to deal with it. Afterwards, Section 4.3 discusses how DeepRED and future deep neural network rule extraction algorithms can be utilized to explain single neurons in a DNN.

4.1 DeepRED – Deep Neural Network Rule Extraction via Decision Tree Induction

As one promising method, we take a closer look at the applicability of CRED (cf. Section 3.2.4) to deep neural networks. We don't expect Sato and Tsukimoto's algorithm to be very fast, but on the other hand it could be more precise than pedagogical approaches like RxREN. And, what might be even more valuable, in theory CRED should be able to learn complex problems like the XOR problem (cf. Section 5.1.3) if it is encoded in a neural network's inner structure.

Since CRED is a decompositional rule extraction approach, it's not possible to apply the original implementation directly to neural networks with multiple layers of hidden neurons. However, the algorithmic approach can be extended relatively easily by deriving additional decision trees and intermediate rules for every supplementary hidden layer.

The modified version of the CRED algorithm – we call it DeepRED[3] – starts just like the original implementation by using C4.5 to create decision trees consisting of split points on the activation values of the last hidden layer's neurons and the regarding classifications in the trees' leaves. To exemplify DeepRED's approach we consider, without loss of generality, a neural network with k hidden layers $h_1, \ldots, h_k$. As a result of the first step we get a decision tree that describes the output layer o by split points on values regarding h_k. We call the respective rule set $R_{h_k \to o}$. The data to run C4.5 on is generated by feeding the training data to the NN and recording the outputs of the hidden neurons.

In the next step, in contrast to the algorithm presented by Sato and Tsukimoto, we don't directly refer to the input layer, but instead process the next shallower hidden layer, i.e. h_{k-1}. For every term present in one of the rules in $R_{h_k \to o}$, we need to apply the C4.5 algorithm to find decision trees that describe layer h_k by means of h_{k-1} and can be transformed to the rule set $R_{h_{k-1} \to h_k}$. Just like in the CRED example presented in Section 3.2.4, the terms in $R_{h_k \to o}$ are directly used to differentiate positive and negative examples for the regarding C4.5 runs.

We proceed in the same manner until we arrive at decision trees/rules that describe terms in the first hidden layer h_1 by terms consisting of the original inputs to the neural network, i.e. $R_{i \to h_1}$. Figure 4.1 provides a pseudo code for DeepRED. Here we see that the proposed algorithm also implements a proce-

[3] DeepRED stands for deep neural network rule extraction via decision tree induction.

```
Input: Neural network NN, training examples x
Output: Set of rules rules

activationValues = getHiddenActivationValues(NN,x)
activationValues(outputLayer) = NN(x)                          // one-hot encoded
foreach currentOutput ∈ outputNeurons do
    // intermediateTerms stores terms that describe higher-level terms
    intermediateTerms(outputLayer, 0) = currentOutput > 0.5
                        // currentOutput is the class of interest, 0 used as dummy
    foreach currentLayer ∈ hiddenLayersDescending(NN) do
        foreach term ∈ intermediateTerms(currentLayer+1) do
            if treeAlreadyExtractedFor(term, currentLayer+1) then
                intermediateTerms(currentLayer, term) = copyTermsFor(term, currentLayer+1)
            else
                intermediateTerms(currentLayer, term) = C4.5(activationValues(currentLayer),
                activationValues(currentLayer+1), term)
                        // Describe term in next deeper layer by terms of current layer
            end
        end
    end
    while getNumberOfLayers(intermediateTerms) > 2 do
        intermediateTerms = mergeIntermediateTerms(getNumberOfLayers(intermediateTerms),
        getNumberOfLayers(intermediateTerms)-1)
        intermediateTerms = deleteUnsatisfiableTerms(intermediateTerms)
        intermediateTerms = deleteRedundantTerms(intermediateTerms)
    end
    rules[] = intermediateTerms2Rules(intermediateTerms)
                // Describes currentOutput by rules consisting of input neuron splits
    delete(intermediateTerms)
end
```

Figure 4.1: Pseudo code of DeepRED.

dure to prevent itself from performing redundant runs of C4.5. If a decision tree to describe a term has already been extracted, it simply copies the results available.

Now we have rule sets that describe each layer of the neural network by their respective preceding layer, i.e. we have the sets of intermediate rules $R_{i\to h_1}, R_{h_1\to h_2}, \ldots, R_{h_{k-1}\to h_k}$, and $R_{h_k\to o}$. To get a rule set $R_{i\to o}$ that describes the NN's outputs by its inputs, these rules need to be merged. This merging process proceeds layer-wise. First, DeepRED substitutes the terms in $R_{h_k\to o}$ by the regarding rules in $R_{h_{k-1}\to h_k}$ to get the rule set $R_{h_{k-1}\to h_o}$. As mentioned in Figure 4.1, unsatisfiable intermediate rules as well as redundant terms are deleted. This happens to reduce computation time and memory usage drastically.

Next, we merge the rule sets $R_{h_{k-1} \to o}$ and $R_{h_{k-2} \to h_{k-1}}$. Step by step we go through all layers until we arrive at rules that describe the classification/outputs according to the inputs to the neural network, which is the result we were looking for.

One way to facilitate the rule extraction process is to prune NN components that can be considered as less important for the classification. While the authors of CRED did not report on any of these approaches, other researchers have invented several techniques that help to extract comprehensible rules. In the subsequent sections, we take a closer look at the pruning approaches implemented by FERNN and RxREN, respectively.

4.1.1 Neuron and Connection Pruning using FERNN's Approach

The idea of FERNN (cf. Section 3.4.2) to find and remove both probably insignificant nodes as well as rather insignificant neuron inputs makes the algorithm an interesting approach. These pruning mechanisms might be able to separate important elements inside a deep neural network from less important information. This is especially true when a DNN's structure isn't the most efficient one possible, which isn't unlikely because in straight-forward neural network training no structure optimization is being applied (cf. Section 2.3).

Unfortunately it is not possible to apply FERNN and its pruning techniques to a DNN in the form proposed by Setiono and Leow. To make the algorithm applicable to neural networks with more than one hidden layer, some modifications need to be implemented first. The original proposal suggests to run the C4.5 algorithm on the outputs of the hidden neurons (hidden outputs), that are extracted by feeding (training) examples to the network. Only those hidden neurons are considered to be relevant, that happen to appear in at least one resulting decision tree split point. Having a DNN and therefore multiple hidden layers, this step needs to be modified.

One way to adjust this step to deep neural networks could be to simply feed the hidden outputs of all hidden layers at once to the C4.5 algorithm. Since the aim of the regarding step is to identify important hidden neurons (in relation to the network's output), we assume that it is more likely for a neuron in a deeper layer to be important than for a neuron in a shallower layer. The reason for this assumption is that the deepest hidden layer is the only one directly connected to the output layer and therefore tends to represent features that are more elaborated and hence more important for the classification. A first test on a two-hidden-layer neural network using this approach affirmed our assumption.

A second approach to handle the multiple sets of hidden outputs is, as a first step, to only feed the outputs of the deepest hidden layer to the C4.5 algorithm. Having identified the relevant nodes in this layer, we proceed in a DeepRED fashion (cf. Section 4.1) to find the relevant neurons in the next shallower layer: For every split point found by C4.5 we create an artificial dataset consisting of the hidden outputs for the shallower layer (copied from the extracted set of hidden outputs) and a boolean value specifying if the condition as stated in the split point is true for the regarding instance. Feeding these data to C4.5 brings us one step closer to the most shallow hidden layer. We repeat this step for every split point on each hidden layer.

This second approach does not only proceed in a DeepRED fashion. In fact, DeepRED intrinsically implements exactly this pruning technique. By only extracting intermediate rules for neurons that are present in a decision tree describing the subsequent layer, all other neurons are ignored in the further process.

For every relevant hidden neuron, FERNN also tries to identify the relevant input connections considering their weights. Unfortunately our experiments show that, for sufficiently large datasets, it isn't possible to identify any irrelevant inputs for most of the hidden neurons. The originally proposed criterion to define irrelevant connections seems not to be applicable to deep neural networks. The fact that we cannot noticeably reduce the number of relevant connections leads us to our conclusion not to extend DeepRED with FERNN's pruning approach.

However, we want to notice that the hidden neuron pruning approach implemented by FERNN could help reducing the complexity of rule extraction algorithms where this technique, different to CRED and DeepRED, isn't intrinsically implemented.

4.1.2 Input Pruning using RxREN's Approach

The RxREN algorithm (cf. Section 3.3.3) implements another interesting pruning mechanism. Since NNs are reported to perform well when having noisy data, the input pruning approach used by RxREN might be able to find the relevant attributes to solve a classification task, while at the same time reducing the extracted rules' potential complexity.

As stated in Section 3.3.3, the input pruning proceeds by testing the neural network's performance while ignoring individual attributes. Those inputs that where not necessary to still produce an acceptable classification performance are getting pruned. Our experiments with the RxREN pruning approach can be found in Section 5.2.2. Since the results are promising, in the same section, we propose a variant of DeepRED using RxREN pruning.

Just like the FERNN pruning approach presented in the preceding section, RxREN pruning can also be applied to an arbitrary neural network rule extraction algorithm. Future methods could profit from a lower number of relevant inputs which is likely to facilitate the extraction process. This holds true for decompositional, pedagogical, and eclectic approaches.

4.2 Experiences with Explaining Deep Neural Networks

As mentioned earlier in this chapter, the problem of extracting rules from deep neural networks hasn't been studied very extensively so far. As also stated, the task can be more challenging than the extraction process from one-hidden-layer neural networks. In this section we want to summarize the experiences that we have gathered while working with DeepRED, FERNN and RxREN. The aim of this section is to give an overview of the challenges mentioned as well as how DeepRED tries to handle them.

4.2.1 More Hidden Layers

For pedagogical approaches that don't consider the neural network structure, the amount of hidden layers is not a very important number. On the other hand, for decompositional approaches an additional hidden layer can greatly complicate the rule extraction process. While working with DeepRED, this problem occurred several times. Especially, the increasing number of intermediate rules that describe a neurons behaviour on the basis of the preceding layer is very challenging. In a worst case scenario, this number can rise exponentially – and so do the memory demands and the computational effort to process the intermediate data.

To overcome this issue, DeepRED implements procedures to delete unsatisfiable rules and redundant terms. This indeed helps DeepRED to extract rules from deep networks, where it wasn't feasible before, having only limited memory and time resources. But, as we present in our evaluation, many two-hidden-layer neural networks still cannot be processed due to the fact that too many intermediate terms and rules are created. We shortly discuss this issue and possible solutions in Section 5.5.

Another challenge with decompositional approaches and deep neural networks arises when an algorithm is extracting approximated rules from single neurons. For instance, DeepRED uses the C4.5 algorithm which is depending on parameters that decide when to stop finding further split points. Hence, DeepRED is also approximating the function, a neuron is representing. However, with multiple layers, there is the problem that "approximate rule extraction methods can suffer from cumulative errors" [Thrun, 1995] which might result in low-quality rules. For this issue, Chapter 5 provides interesting figures, too.

4.2.2 More Complex Concepts

Especially for pedagogical algorithms it can be very challenging to model the potentially complex concepts an output neuron is representing. The probability that more complex concepts are described in the output neurons is increasing with a larger amount of hidden neurons. Rather simple rule generation mechanisms can result in too simple rules that don't model the neural network's behaviour accurately enough.

Decompositional approaches have the advantage of using a divide-and-conquer strategy to break the model a neural network is representing down into smaller pieces by making use of its inner structure. This helps such algorithms to map the NN behaviour to rules. Since neurons in a neural network in general don't realize e.g. split points like C4.5 does, a description using rules can suffer from bad approximation. Let's consider the simple function $i_1 \geq i_2$ with $i_1, i_2 \in \mathbb{R}$. A perceptron (cf. Section 2.3), the simplest neural network possible, can easily model this function[4] while in the worst case C4.5 needs to find a split point for every single training example.

This shows that the expressive power of e.g. decision trees (and the rules that can be extracted from them) does not necessarily meet the characteristics a neuron is representing. In some cases, rules of this

[4] The perceptron would consist of an input layer with the two inputs i_1 and i_2. If the relevant output has a threshold of zero, the weight from i_1 to this output would be 1 while the respective weight from i_2 would be -1.

form might be able to approximate the function a neuron is modelling much better. But depending on the learning problem and the depth of the neural network, more layers can mean more complex concepts that, if at all possible, achieve acceptable accuracy/fidelity rates only with very long rules that tend to be less comprehensible than shorter rules.

To overcome this limits, some authors choose to implement other rule types, like M-of-N or fuzzy rules (cf. Section 2.2) that might be able to approximate a neuron's function better because their general approach is more similar to that of a neuron. On the other hand, these different forms of rules might be less comprehensible ones. A variant of DeepRED could also be developed, where C4.5 could be replaced by other rule extraction mechanisms, e.g. RIPPER (cf. Section 2.2.3).

4.2.3 Large Sets of Training Examples

To train a neural network, large training sets are necessary. This becomes even more important for every additional layer of hidden neurons. So, in general, for a trained deep neural network a relatively large set of training instances should be available.

But more training examples does not necessarily mean an easier rule extraction task. E.g. the runtime of DeepRED strongly depends on the number of training examples available, i.e. more instances mean a longer runtime. As our experiments in Chapter 5 show, DeepRED needs a minimum number of training instances to produce the best fidelity rates. However, some rule extraction tasks with a large set of training instances cannot be successfully processed due to too high memory requirements. Since every run of C4.5 considers all the training examples available, this effect isn't unexpected. Therefore, for the extraction of rules from deep neural networks, a good tradeoff between more accurate rules and the overall feasibility to extract rules needs to be found for environments with limited time and memory resources.

Pedagogical approaches, however, normally benefit from a more extensive training set. Since they don't consider a neural network's inner structure, the problem's complexity in general only grows linearly with the training set's size, while the achieved fidelity rates get better.

4.3 Explaining Neurons in Deep Neural Networks

For deep neural networks, the rule extraction problem can be ambiguous. It might not only be interesting to explain an output's behaviour, but also to describe the behaviour of a neuron in a hidden layer – or its importance for an output neuron. In this section, we give an overview of how decompositional rule extraction algorithms like DeepRED or pedagogical approaches can be used to analyse and explain the behaviour of a DNN.

"It is well-known that deep neural networks are forming an efficient internal representation of the learning problem" [Montavon, 2013]. But even though the idea of neural networks isn't entirely new, often "it is unclear how this efficient representation is distributed layer-wise, and how it arises from learning" [Montavon, 2013]. Explaining single neurons with rules can help to analyse the learning process.

An interesting task in this context is to answer the question, what neurons are worth to be explained. E.g. Decloedt et al. did research on the topic of selecting "elements representing more significant network knowledge" [Decloedt et al., 1996]. Although the results of this research area might be valuable for some applications of rule extraction from neurons, we won't discuss this subject more extensively in this thesis.

As an alternative solution to analyse the way, a neural network realizes its classifications, one could also think of a DeepRED variant that produces rules with an explicit notation of how the NN realizes them. Our proposed algorithm acquires the information necessary (in the form of intermediate rules) but discards them. For instance, using brackets in the finally extracted rules could lead to a better understanding of how a decision is formed, i.e. what functions the relevant neurons realize. This might also help to identify hidden features. However, we leave this modification for future research due to two reasons: First, it is probable that such an approach would produce longer and more rules since the deletion of redundant terms and rules in some cases cannot be performed without loss of information. And second, for each additional layer, the rules become more complex, which in the end makes it difficult to really understand DNNs.

Hence, in the remainder of this section, we focus on ways to analyse a neural network with an arbitrary DNN rule extraction algorithm and shortly introduce the following tasks: How can neurons be described on the basis of their inputs? How do hidden neurons depend on the neural network's input? What is a hidden neuron's effect on single output neurons?

4.3.1 Explaining Hidden Neurons on the basis of their Inputs

In some cases, it might be interesting to know what function a single neuron is realizing. For instance, when it is already known what models the neuron's inputs are representing, knowing the neuron's immediate function can help to easily describe its function in the context of the whole network. As an example, think of a neuron where it is known, that its two inputs are realizing the functions $i_1 \wedge i_2$ and $\neg i_1 \wedge \neg i_2$, respectively. If the regarding weights to the neuron of interest both are -7 while the bias is 3 and the threshold is zero, it is trivial to conclude that the function represented by the neuron is an XOR of i_1 and i_2.

To extract rules that describe a neuron's behaviour relatively to its immediate inputs, an arbitrary pedagogical extraction algorithm can be applied. We only need to modify the process such that, instead of passing the neural network's black box function $NN()$, we only pass the neuron's raw function to the pedagogical approach, for example $\sigma(\sum_j w_j i_j)$. The training set needs to be substituted by the output values of the neuron's input neurons. This is achieved by feeding the original training data to the network and memorising the needed values directly produced by the regarding neurons.

Alternatively, we could use the methods of any decompositional approach. As stated in Section 3.2, these algorithms usually implement methods to analyse a neuron and to form rules to mimic the behaviour of this unit. In the case of DeepRED, C4.5 is utilized – but many more approaches are conceivable. Hence, the intermediate rule extraction technique implemented by a decompositional algorithm can simply be used to explain a hidden neuron's behaviour on the basis of its direct inputs.

4.3.2 Explaining Hidden Neurons by Shallower Layers

Another task of interest could be the description of a hidden neuron by means of neurons in shallower hidden layers or the input layer. Explaining the function of such a neuron could help to analyse the possibly more complex concept it is modelling. For instance, in neural network learning using the MNIST dataset (cf. Section 5.1.1), many researchers use visualizations to give insights into the concepts individual hidden neurons are representing [Duch et al., 2004]. For other learning problems, rules to describe these concepts might be worthwhile.

The extraction of rules in this context can also be realized by pedagogical algorithms and decompositional approaches that can handle deep neural networks. The modification of the input parameters to the methods is similar for both kinds of rule extraction algorithms. If the inputs of interest are from a hidden layer, the training data need to be converted in the same way as described in the preceding section. To get the neural network or the neural network function according to Figure 3.1 and Section 3.3, respectively, we simply cut the rear of the network such that the hidden neuron that we want to explain becomes an output neuron. E.g., in the network depicted in Figure 4.2, if we want to extract rules for a neuron in *hidden layer 3*, we would delete the *output layer*.

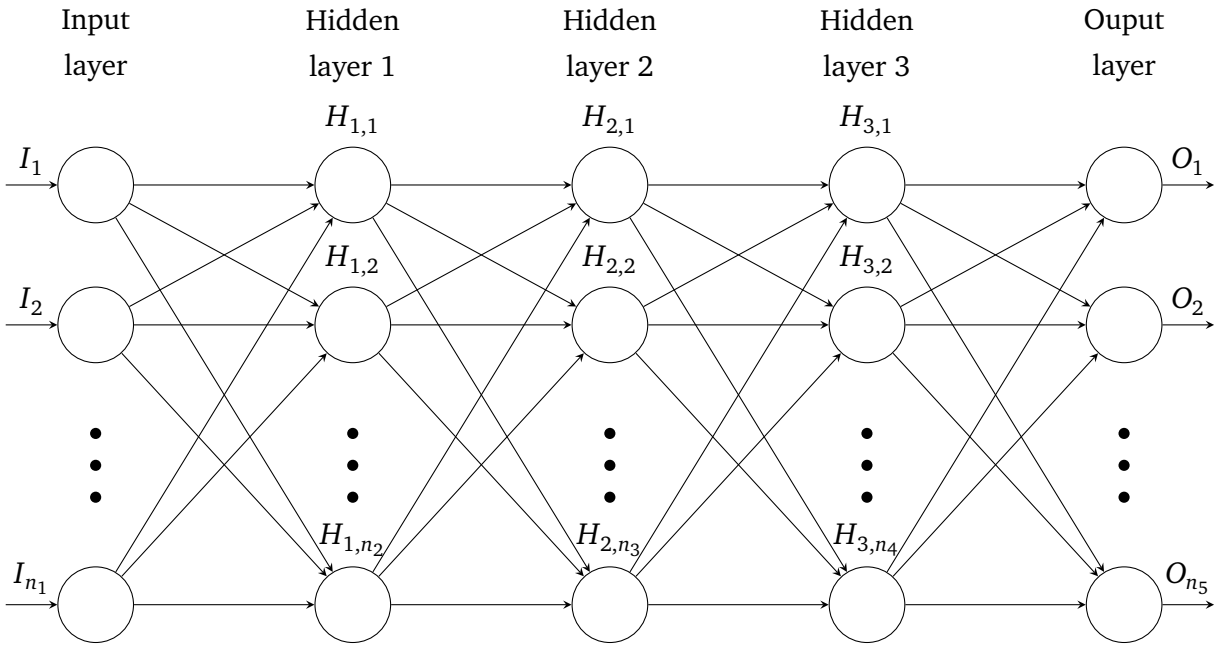


Figure 4.2: A three-hidden-layer neural network can be modified to extract rules that describe neurons of an arbitrary layer by means of neurons in an arbitrary shallower level.

Likewise, if we are interested in a description by means of neurons in *hidden layer 1*, we cut the network's layers that are shallower than the inputs of interest, i.e. the input layer. Feeding these data to a rule extraction algorithm results in the extraction of the rules wanted.

4.3.3 Explaining a Hidden Neuron's Effects on Deeper Neurons

A last interesting problem, we want to introduce here, is to explain the effect of a hidden neuron on neurons in deeper layers of the NN. While analysing a trained network, it might be helpful to know how an individual neuron in a hidden layer affects the output or a neuron in between. Rules can offer an insight by using the approach described in the preceding section. Considering the neural network in Figure 4.2, if we want to analyse the impact of a neuron in *hidden layer 2* on a neuron in the *output layer*, we cut the NN's *input layer* as well as *hidden layer 1*. As a result we get rules that describe the output neurons by means of the neurons in *hidden layer 2*.

4.4 Concluding Remarks

In Chapter 4 we took a closer look at the problem of extracting rules from deep neural networks. We introduced a new algorithm to perform this task, reported our experiences with the challenges of DNNs, and presented general approaches to analyse a deep neural network with an arbitrary DNN rule extraction algorithm.

Since our state-of-the-art analysis in Chapter 3 has shown that existing approaches usually cannot handle deep neural networks, in Section 4.1 we proposed DeepRED which can extract rules from DNNs. We also analysed how pruning can help DeepRED to extract more comprehensible rule sets.

Section 4.2 summarized our experiences with DeepRED regarding three DNN challenges, i.e. more hidden layers, more complex concepts, and possible problems with large training sets.

Afterwards, Section 4.3 provided instructions on how to use an arbitrary deep neural network rule extraction algorithm to analyse single neurons in a DNN. We presented general approaches to explain a neuron's behaviour according to its inputs or shallower neurons as well as a neuron's importance for deeper layers.

The proposed algorithm should be able to successfully perform the task of extracting rules from a DNN. To prove this and to compare DeepRED's performance to a baseline approach, Chapter 5 provides an evaluation of our algorithm.

5 Evaluation

In the preceding chapter, we have proposed DeepRED – a new algorithm that can extract rules from deep neural networks. In Chapter 5, we present the algorithm's evaluation. Here, we want to learn about the strengths and weaknesses of DeepRED when applying it to different kinds of data.

In Section 5.1 we introduce the data the evaluation is based on. Afterwards, Section 5.2 summarizes the parameters of DeepRED and presents the baseline algorithm. The measures we use to compare our approach with the baseline are provided by Section 5.3.

The experiments and our expectations are described in Section 5.4, before we present and discuss the evaluation results more extensively in Section 5.5.

5.1 Deep Neural Networks for Evaluation

To evaluate DeepRED, we need the regarding input data. Figure 4.1 tells us, that we need a trained neural network as well as a training set that can be classified by this NN. Unfortunately, it is not trivial to find already trained neural networks in literature or online. Although NN research is a lot about training, usually only network structures, training methods and performances are reported, while specific weights are not of particular interest for other researchers.

This is why we need to train neural networks by ourselves. Since we use MATLAB as the main development environment, we choose to utilize the *DeepLearnToolbox* that also provides learning mechanisms to more efficiently train deep neural networks [Palm, 2012]. In the remainder of this section, we present the datasets that are the basis for the regarding neural networks as well as the trained networks. Table 5.1 summarizes the DNNs used for this evaluation.

Please note that, in the remainder of this evaluation, we use the dataset names for both, the datasets and the neural networks trained on these data.

Table 5.1: Overview of deep neural networks used for evaluation, including the characteristics of the original data the NNs were trained on.

	#attributes	#training ex.	#test ex.	NN structure	acc(training)	acc(test)
MNIST	784	12056	2195	784-10-5-2	99.6%	98.8%
letter	16	1239	438	16-40-30-26	96.9%	97.3%
artif-I	5	20000	10000	5-10-5-2	99.5%	99.4%
artif-II	5	3348	1652	5-10-5-2	99.4%	99.0%
XOR	8	150	106	8-8-4-4-2-2-2	100%	100%

5.1.1 The MNIST Database of Handwritten Digits

MNIST is a popular dataset used by many researchers to evaluate different machine learning algorithms [LeCun et al., 1998]. As depicted in Figure 2.1 and mentioned in Section 2.1, the MNIST database describes handwritten digits from zero to nine by a number of 784 greyscale attributes, i.e. values from zero to 255.

While the original dataset consists of 60000 training examples and the test set of 10000 instances, we modify the database to facilitate the problem to only distinguish instances of the first class from examples of other classes. As shown in Table 5.1, the smaller dataset consists of 12056 training examples (6133 belong to class 1[1]) and 2195 test examples (1207 belong to class 1). We trained a deep neural network with four layers on the 12056 training examples to classify them into class 1 (first output neuron) and others (second output neuron).

You might wonder why we use two output neurons for a binary classification problem. This is due to the way, we use a neural network to predict an instance's class. As defined in Section 2.3, we use the output neuron with the maximum activation value to determine the class. If there would be only one single output neuron, the NN would always predict class 1.

For more details on the neural network learned for the MNIST dataset, please refer to Table 5.1.

5.1.2 The Letter Recognition Dataset

To find a second suitable real-world dataset, we have searched the *UCI Machine Learning Repository* [Lichman, 2013]. With the *Letter Recognition Dataset* we could find a problem that is very similar to MNIST – the task is to classify letters [Frey and Slate, 1991]. However, the original pixel graphics have been transformed by the authors to a number of 16 attributes. They represent special characteristics using integer values from zero to 15 for each attribute.

We have trained a two-hidden-layer neural network using 15000 instances, while 5000 examples are used as the test set. The DNN's characteristics are provided by Table 5.1. Afterwards, we have cut the training set to 585 examples of the first class (represented by the first output neuron) and 654 of other classes[2]. The test set consists of 204 examples of the first class and 234 instances that don't belong to class 1. Please note, that the accuracy rates in Table 5.1 are based on this smaller dataset – that we abbreviate with the term *letter*.

5.1.3 The XOR Problem

The *XOR* problem – short for *exclusive or* – is well-known to be a difficult problem for many rule learning algorithms. We formalise XOR as a two-class problem on an arbitrary number of binary attributes,

[1] Class 1 contains only examples that do not represent zeros, while class 2 only contains handwritten zeros. The original MNIST dataset provides 5923 zero examples, so for class 1 we randomly selected approximately the same number of non-zeros. The procedure to select the test examples was similar.

[2] Similar to the MNIST selection process, the original letter dataset provides 585 examples of the relevant class, while we selected approximately the same number of instances from other classes. The procedure for the test set was the same.

i.e. zero or one. Only those instances belong to the first class where the number of attributes, that are set to one, is odd. In our concrete dataset we have eight attributes.

This results in a database with a maximum number of 256 distinct instances. We divide the dataset into a training set of 150 examples and a test set with 106 instances. There are 73 examples in the training set that belong to the first class and 55 test examples that are of class 1.

Since training a neural network to solve the XOR problem isn't trivial as well, we manually constructed a deep neural network with a total number of seven layers. The first output neuron is meant to fire if an example belongs to the first class. The resulting DNN is depicted in Figure 5.1. The weights are shown at the regarding connections, the numbers in the neurons are the respective biases.

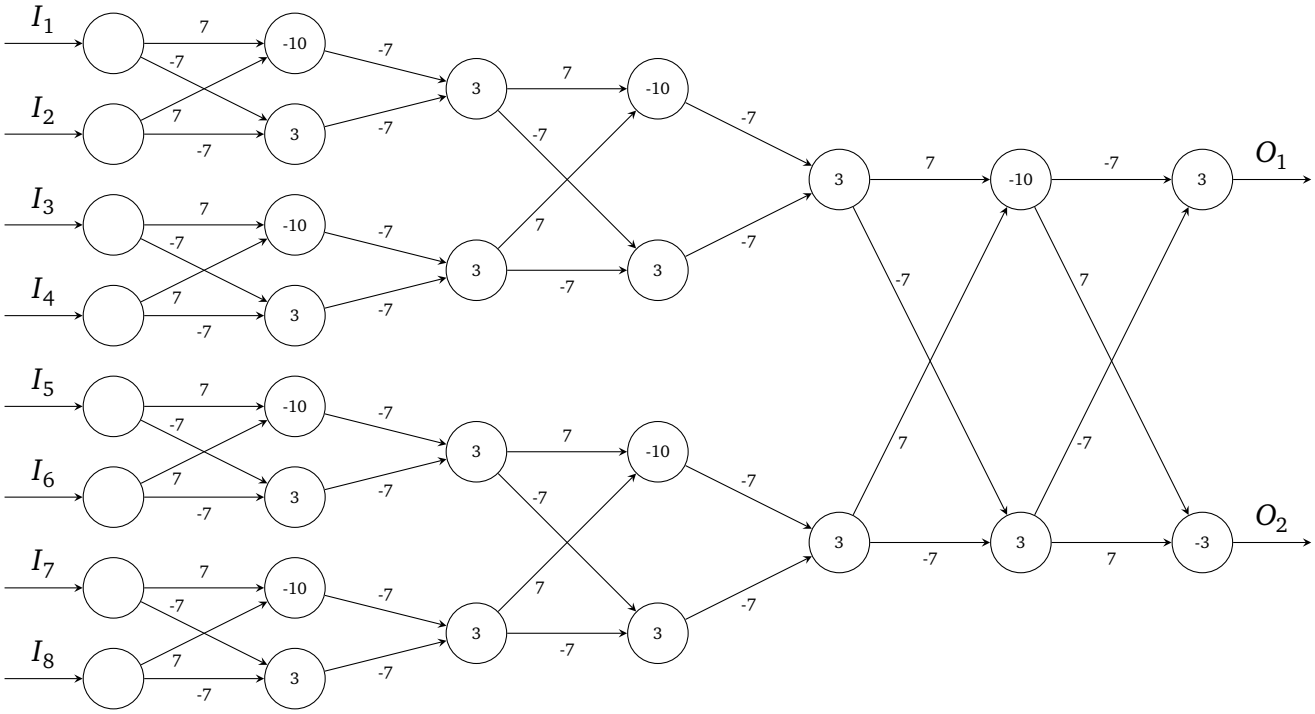


Figure 5.1: A deep neural network solving the XOR problem with eight attributes.

The depicted neural network uses several XOR functions on two inputs to create an XOR with eight attributes. The general structure that, for instance, can be found consisting of I_1, I_2, and the connected neurons in the first and second hidden layer can be found three more times in the first three layers, twice in the layers three to five and once in the last three layers. This two-input XOR function is realized by $\neg(I_1 \wedge I_2) \wedge \neg(\neg I_1 \wedge \neg I_2)$. The classification accuracy of this neural network is 100% for both, the training set and the test set.

5.1.4 Artificial Dataset I

In addition to the datasets and neural networks presented, we have created two artificial databases to compare DeepRED with the baseline. These two datasets have been created, because knowing the true models can help qualitatively analysing the results of the extracted rules. Since MNIST and letter are real-world datasets, i.e. the true models are not known, and a neural network for the XOR problem

couldn't be trained automatically, we want to use the outcomes using our artificially created datasets if a better understanding of the true models is necessary.

Both databases comprise examples with five attributes. The first input x_1 can take on values in $\{0, 0.5, 1\}$, the second x_2 can be one of the values in $\{0, 0.25, 0.5, 0.75, 1\}$, while x_3, x_4, and x_5 can take on a random value in $[0, 1]$, each.

The first artificial dataset randomly generated using the mentioned constraints is called *artif-I*. The model that determines the instances' classes is a rule set consisting of the following rules: IF $x_1 = x_2$ THEN $\boldsymbol{y} = y_2$, IF $x_1 > x_2$ AND $x_3 > 0.4$ THEN $\boldsymbol{y} = y_2$ IF $x_3 > x_4$ AND $x_4 > x_5$ AND $x_2 > 0$ THEN $\boldsymbol{y} = y_2$, ELSE $\boldsymbol{y} = y_1$.

A challenging characteristic of the artif-I dataset is that it contains greater-than relations on the real-valued attributes x_3, x_4, and x_5. These functions cannot easily be modelled by decision trees.

With 20000 training examples and 10000 test examples, artif-I is the largest dataset we use for our evaluation. While 9911 training instances belong to class y_1, there are 4898 examples of class y_1 in the test set. The two-hidden-layer neural network learned from this data is able to correctly classify 99.4% percent of the test set and 99.5% of the training set. The first output is realizing the class y_1.

5.1.5 Artificial Dataset II

The second artificial dataset, *artif-II*, is generated using the same constraints as mentioned in the preceding section. However, the model to determine which class an example belongs to is different. The model consists of the following rules: IF $x_1 = x_2$ THEN $y = \boldsymbol{y}_1$, IF $x_1 > x_2$ AND $x_3 > 0.4$ THEN $\boldsymbol{y} = y_1$, IF $x_5 > 0.8$ THEN $\boldsymbol{y} = y_1$, ELSE $\boldsymbol{y} = y_2$. Please note, that x_4 has no effect on an instance's class. This might be challenging for rule extraction algorithms.

The resulting dataset consists of 3348 training examples (where 1842 belong to y_1) and 1652 test instances (where 944 belong to y_1). Using the training data we trained a neural network with the same structure as for artif-I, again the first output is meant to realize y_1. The NN's performance is provided by Table 5.1.

5.2 Rule Extraction Algorithms

To get an idea of how well the proposed algorithm performs the task of extracting rules from deep neural networks, in this evaluation we compare two variants of DeepRED – a version without and a version with RxREN pruning – with a pedagogical baseline. In this section, we briefly introduce the three algorithms and their parameters.

5.2.1 DeepRED

Section 4.1 has already provided the main procedure and details of how DeepRED extracts rules from a deep neural network. However, there are two parameters that can be passed to DeepRED to change the behaviour of the integrated C4.5 algorithms.

Table 5.2: Error rates and neural network characteristics after RxREN pruning.

RxREN pruning intensity	training set error	test set error	number of input attributes
No RxREN pruning	0.0089	0.0467	784
Pruning intensity 1	0.0088	0.0463	657
Pruning intensity 2	0.0132	0.0495	580
Pruning intensity 3	0.0437	0.0684	465
Pruning intensity 4	0.1162	0.1245	355

Both parameters control when C4.5 stops further growing a decision tree. The first one – *class dominance* – is a threshold that considers the classes of the current data base. If the percentage of examples that belong to the most frequent class exceeds the value in the class dominance parameter, this class is getting predicted instead of further dividing the data base.

The second parameter is the minimum *data base size*. This value tries to stop C4.5 further growing the decision tree when there is not enough data available to base dividing steps on. This parameter takes the number of training examples available in the first step as a reference value and defines a percentage of this size as the minimum data base size. If for the current step, there are less examples available than the parameter requires, C4.5 produces a leaf with the most frequent class at this point.

For our experiments, we also adjust C4.5 to only perform binary splits and to produce a maximum decision tree depth of ten.

5.2.2 DeepRED with RxREN Pruning

As an extension of DeepRED, we want to evaluate how well DeepRED with RxREN pruning works. To test the capabilities of the RxREN pruning approach, we performed a pre-experiment on a two-hidden-layer neural network describing the MNIST dataset (cf. Section 5.1.1). This experiment resulted in a pruning rate of 26% which means that, according to the pruning mechanism, a number of 204 out of 784 input attributes can be considered as insignificant for the classification, while reducing the pruned neural network's accuracy by less than 0.3 percentage points. Especially interesting is the fact that the network's error rate is slightly decreasing when pruning the 127 most insignificant of the 784 inputs[3]. Figure 5.2 visualizes the reported results as well as the results for different RxREN pruning thresholds, Table 5.2 provides the regarding numbers.

While this extension of DeepRED also has the freedom to set different C4.5 parameters, RxREN pruning offers a further parameter, namely the maximum *accuracy decrease*. Step by step, the technique prunes the inputs that are considered as the least significant ones, as long as the neural network's accuracy does not drop below the decrease allowed by the parameter.

Please note that, if there is more than one input with the exact same lowest significance value, the technique prunes all these inputs at the same time. If the threshold value for the maximum accuracy

[3] Please consider that RxREN pruning just sets the weights from each insignificant attribute to all neurons in the first hidden layer to zero. With a retraining step we would have assumed the effect reported, but in this case no retraining or other adjustments take place.

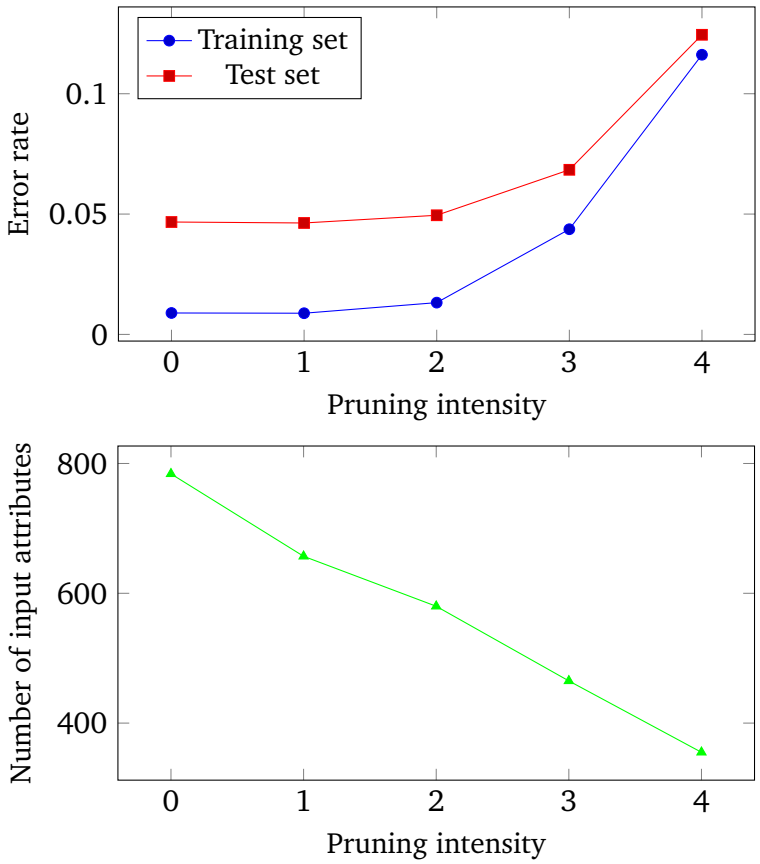


Figure 5.2: Error rates and neural network characteristics after RxREN pruning.

decrease isn't reached, the group of inputs with the next-lowest significance value gets pruned. We refer to such a step by calling it pruning intensity[4], while in Figure 5.2 pruning intensity 0 means that no pruning has been applied.

5.2.3 C4.5 as a Pedagogical Baseline

As a pedagogical rule extraction baseline we choose to use the well-known C4.5 algorithm (cf. Section 2.2.3), that is also used by DeepRED. This means, C4.5 is provided with the training examples as well as the neural network's classification for these instances. Using these values, the baseline extracts a decision tree that is transformed to a rule set.

The baseline can be modified by changing the same values as presented in Section 5.2.1, i.e. a parameter for the class dominance as well as a value for the minimum data base size. We also let the baseline only perform binary splits. Just like in DeepRED, only decision trees with a maximum depth of ten are produced by the baseline.

[4] As an alternative to the maximum accuracy decrease parameter, one could also use the pruning intensity as a paramter. However, since finding a good value would be more complicated with such a setting, we only consider the decrease value as a parameter to RxREN pruning.

At this point we want to clarify, that no rule rewriting, rule pruning or any other rule optimization mechanisms are implemented by the DeepRED variants[5] or the baseline. Furthermore, there have not been applied any post-processing steps to modify the extracted rule sets. Please consider this fact when reviewing the evaluation outcomes.

5.3 Evaluation Measures

To compare the results of the algorithms presented in Section 5.2 on the trained deep neural networks presented in Section 5.1, we derive two central measures from the extracted rule sets, that are used to assess their quality.

As discussed earlier in this thesis (cf. Chapter 3), we measure the comprehensibility merely by the number of terms in the extracted rule set. For every rule in the rule set, we count the terms (cf. Section 2.2) and sum up all these numbers. We don't need to consider the default rule, since there are no terms present in the condition. This value can be any non-negative integer – the lower the number of rules, the better the comprehensibility.

To quantify how well the extracted rules can mimic the behaviour of the deep neural networks, we use the so-called fidelity. As mentioned in Chapter 3, the fidelity measures the accuracy based on the neural networks output. We calculate this measure by using both the neural network NN and the extracted rule set $rules$ to classify the examples $\boldsymbol{x}_i$ in the test set $D_{test} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_t, \boldsymbol{y}_t)\}$.

$$fidelity(NN, rules, D_{test}) = 1 - \frac{1}{t} \sum_{i=1}^{t} \Delta(NN(\boldsymbol{x}_i), rules(\boldsymbol{x}_i)) \tag{5.1}$$

Again, we see that the instances' true classes have no impact on the fidelity rate of a rule set. The function Δ returns 1 if its two parameters have different values, otherwise 0 is returned. The fidelity can take on values between zero and one (both including) – the higher the fidelity, the better the mimicking performance.

5.4 Evaluation Setup

In this section, we summarize our expectations of how well DeepRED works and where we think, the baseline will outperform the proposed algorithm. We also explain how we set up the experiments to check our expectations. Furthermore, we briefly describe the evaluation hardware settings.

5.4.1 Expectations of DeepRED's Performance

On the one hand, we don't expect DeepRED to be impeccable. We know that there is room for improvements and feature tuning. On the other hand, we believe that there are several tasks, where this version of DeepRED already outperforms simple pedagogical approaches like the baseline used. To analyse the

[5] Except the reported steps to delete redundant terms and unsatisfiable rules.

advantages and disadvantages, we want to conduct experiments that provide us with evidence for our assumptions – or refute them. Our expectations are:

1. *DeepRED is able to extract rules from deep neural networks*.

   As stated in Chapter 4, to the best of our knowledge, no rule extraction algorithm has ever been tested on a deep neural network. We believe that in general the proposed algorithm can manage this task. However, we also expect DeepRED's memory and computation time demands to sometimes make rule extraction processes intractable. We assume that this is especially true for classification problems where an instance is described by a large number of attributes.

2. *DeepRED outperforms the baseline on more complex problems*.

   We believe that our algorithm can profit from a neural network's structure when the training data model a concept that cannot be described/learned by a simple decision tree – including but not restricted to the XOR problem. On the other hand, for decision tree problems, i.e. classification tasks that can more easily be solved by a decision tree, our assumption is that the baseline can outperform DeepRED.

3. *RxREN pruning facilitates the extracted rules*.

   For problems where not all inputs are relevant, we expect the extracted rules to be more comprehensible when RxREN pruning has been applied. Depending on the problem, we even think that the fidelity rates can be improved. However, since the optimal setting of the pruning parameters isn't trivial, the extracted rules probably won't reach the possibly best tradeoff between comprehensibility and fidelity.

4. *DeepRED extracts more accurate rules if more data is available*.

   We believe that the DeepRED algorithm needs a certain amount of data to extract reasonable rules. However, we also think that our algorithm is less dependant on a sufficiently large dataset than C4.5 is, because DeepRED focusses on the setup of the neural network's inner structure using instances, which is a more efficient representation compared to the pedagogical point of view.

On top of these expectations, we hope to find some further indications for areas where DeepRED still needs to be improved. In the subsequent section, we present the experiments conducted to check the validity of the assumptions just presented.

5.4.2 Experiments

On the basis of the data presented in Section 5.1, we want to compare DeepRED with the baseline. As introduced in Section 5.2, we can modify two parameters for the C4.5 algorithm (used by both DeepRED and the baseline) that manage the stopping criteria when building decision trees, i.e. class dominance and data base size. Since an automated adaptation is beyond the scope of this thesis, we decide to compare three different settings for the C4.5 algorithm, that are

1. 92% dominance and $\leq$ 2% size
2. 95% dominance and $\leq$ 1% size
3. 99% dominance and $\leq$ 0% size.

The first setting produces shorter decision trees that don't tend to overfit, the third parameter setting leads to decision trees that describe the training sets in more detail, while the second setting can be seen as a tradeoff of the other two goals.

For the RxREN pruning, we also decide to try three settings. We either don't use RxREN pruning at all, or apply the pruning mechanism to decrease the accuracy rate by up to 5% or 10%, respectively. We are aware that a larger number of settings could be valuable here. But since it is probable that this would result in many experiments with exactly the same outcome, we limit ourselves to the setup presented.

As a last variable, we want to control the number of training examples visible to the rule extraction algorithm. From the original set of training instances we pass random subsets of either 10%, 25%, 50% the size, or all the examples to the regarding approach.

The number of experiments for the proposed algorithm sums up to 180 (or 36 per dataset) while the baseline is executed in 60 experiments (or 12 per dataset) because RxREN pruning never is applied here. Please also note that we set up the experiments to only extract rules for the first output neuron of the regarding DNN. Table 5.3 summarizes the different parameter settings chosen.

Table 5.3: The parameter settings used for the evaluation.

C4.5 parameters	RxREN pruning	Training set
92% / $\leq$ 2%	No pruning	10%
95% / $\leq$ 1%	5%	25%
99% / $\leq$ 0%	10%	50%
		100%

5.4.3 Evaluation Hardware

To conduct our experiments, we had access to the *Lichtenberg High Performance Computer* at TU Darmstadt[6]. Since DeepRED isn't a very efficient algorithm and due to the large number of individual experiments, using the cluster can dramatically reduce the time needed for evaluation.

Each experiment is executed as an individual job, while we set the maximum memory consumption of each job to 10000MB. If the experiment exceeds this limit, it gets aborted. The same holds true for the maximum execution time of 24 hours.

5.5 Results

The execution of the experiments mentioned in the preceding section lead to a good basis to assess DeepRED's performance, its strengths, and its weaknesses. Some results match our assumptions, some

[6] Further information can be found online at www.hhlr.tu-darmstadt.de.

outcomes are partly surprising. We structure the presentation of our evaluation's results using the order in which we described our expectations in Section 5.4.1.

5.5.1 DeepRED can Successfully Extract Rules from Deep Neural Networks

In accordance with expectation 1 mentioned in Section 5.4.1, we first want to analyse if DeepRED is generally able to extract rules from deep neural networks. Table 5.4 gives an overview of the success. For every dataset, we list the number of experiments that were executed[7] as well as the number of successful and aborted attempts. An abortion could either be the case if the experiment exceeds the allocated memory space or if DeepRED needs more than the maximum execution time (according to Section 5.4.3).

Table 5.4: Some experiments with DeepRED couldn't successfully be finished due to too large memory consumption or computation time requirements.

	artif-I	artif-II	letter	MNIST	XOR
Executed	36	36	36	21	12
Successful	11	23	26	4	12
Aborted (memory)	24	13	10	7	0
Aborted (time)	1	0	0	10	0

One positive observation is that there is always at least one parameter setting that makes DeepRED successfully extract rules from each deep neural network tested. Except for the MNIST dataset, this also holds true for every RxREN pruning variant evaluated. On the other hand is the high abortion rate of approximately 46% an indication that the right parameters are important when using DeepRED. As expected, especially for classification problems with a large number of inputs, i.e. MNIST, the success rate is very low[8].

A further investigation to find out at what point in the source code DeepRED exceeds the memory limit shows us that all of the abortions happen during the merging process. Most of the time it occurs while merging the first two layers of intermediate rules, in some cases the second step of the merging process leads to an abortion due to too high memory requirements. We believe that a huge amount of intermediate rules in both layers leads to the production of an even larger number of intermediate rules in the merged layer that exceeds the limits. For future work, one could think of a more elaborate way to make C4.5 stop growing too large trees, when memory problems are foreseeable.

The abortions due to a too long computation time most of the time happen while the extraction of the intermediate rules, i.e. while C4.5 is used to create decision trees. As mentioned earlier, DeepRED has its limits in processing large amounts of data. The time abortions can all be ascribed to large training sets passed to the algorithm. One interesting case is the abortion of DeepRED with 10% of MNIST's training

[7] You might notice that, earlier in this chapter, we mentioned that there are 36 experiments per dataset. However, to avoid sophisticating the outcomes, we discard those experiments where the RxREN pruning results in no pruned inputs at all.

[8] For the MNIST dataset, the settings of the successful experiments were 92% / $\leq$ 2% C4.5 / 5% pruning / 100% data, 92% / $\leq$ 2% / 10% / 50%, 92% / $\leq$ 2% / 10% / 100%, and 95% / $\leq$ 1% / 10% / 100%, respectively.

data, no pruning, and the C4.5 parameters 92% / $\leq$ 2%. Here, the rule extraction actually was possible. However, the calculating of the fidelity of the 275104 extracted rules could not be finished during the 24 hours computation time.

5.5.2 DeepRED can Extract Comprehensible Rules for Complex Problems

To check if our expectation 2 is fulfilled, we want to take a closer took at the results of DeepRED and the baseline on the two artificial datasets artif-I and artif-II. As described in Section 5.1, the data in artif-I model a function that cannot easily be realized by a decision tree because greater-than relations between real-valued inputs are part of it. We consider this a difficult problem for the baseline. The artif-II data, on the other hand, can quite easily be modelled by a decision tree. However, please notice that the functions the deep neural networks have learned are not equivalent to the true models.

Figure 5.3 shows the results[9] of DeepRED and the baseline on artif-I (on the left) and artif-II (on the right). We clearly see that the best DeepRED result outperforms the baseline on the deep neural network representing the artif-I dataset. Especially the comprehensibility of the rules extracted by our approach is much better. It seems like the proposed algorithm could use the inner structure of the DNN to create rules that efficiently model the neural network's behaviour. But, on the other hand, we must also admit that the fidelity rates are not as good as they could be – or as they should be to appropriately explain a neural network.

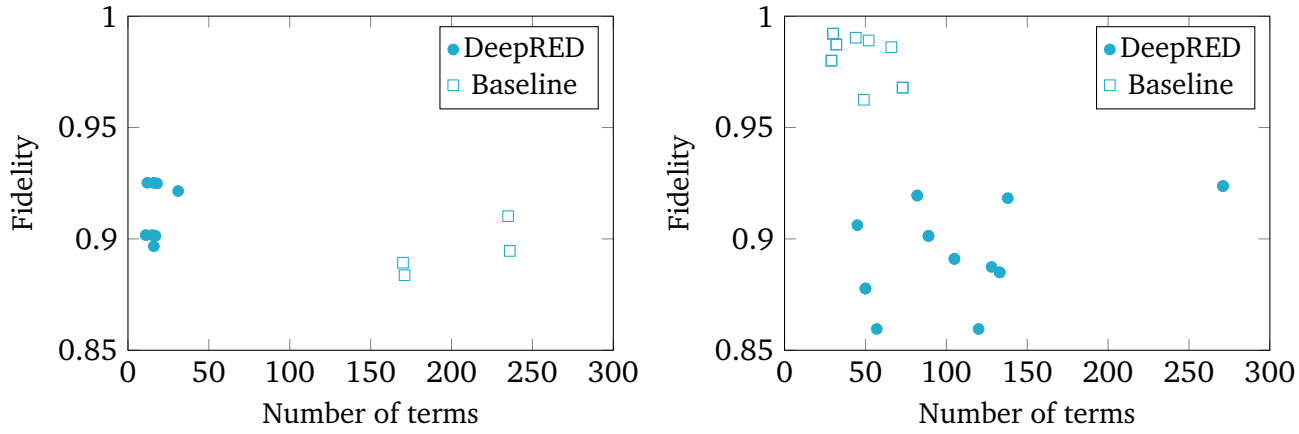


Figure 5.3: Evaluation results for DeepRED and the baseline on artif-I (left) and artif-II (right).

The artif-II dataset gives a different picture: The baseline produces much better results than DeepRED does. Regarding fidelity, the outcomes of our approach are more or less similar to those of the artif-I data. However, the baseline is able to find a more efficient way to model the neural network's outputs – and does so much more accurate. For some reason, the inner structure of the DNN seems to confuse DeepRED such that it cannot manage an extraction of high-quality rules.

Part of our expectation also was, that DeepRED can outperform the baseline by far when extracting rules from an XOR neural network. Figure 5.4 gives an insight into the performance on this task. Please

[9] For reasons of clarity and comprehensibility, we only show the results for the extracted rules that reach a fidelity of at least 85% as well as a maximum of 300 terms.

notice, that compared to Figure 5.3 we needed to adjust the axes to draw a meaningful graph. As we expected it and it is well-known, C4.5 cannot generalize from XOR training examples to correctly classify distinct test instances, i.e. it never gets better than random. However, as we see in the graph, DeepRED is able to use the knowledge embedded in the deep neural network, to extract rules that correctly classify unseen examples. As we will discuss in a moment, DeepRED needs a sufficiently large data basis to perform this task. But if it is given, an error rate of 0% can be achieved.

You might argue that DeepRED isn't extracting comprehensible rules. In fact, rules in the form as we use them are not able to efficiently model the XOR problem. To correctly classify all possible 256 input combinations for this problem, 128 rules with 8 terms each are necessary to cover all 128 positive examples (the default rule covers the negative examples). In sum this leads to a number of 1024 terms in the rule set – which is the number of terms, the most accurate rules extracted by DeepRED have.

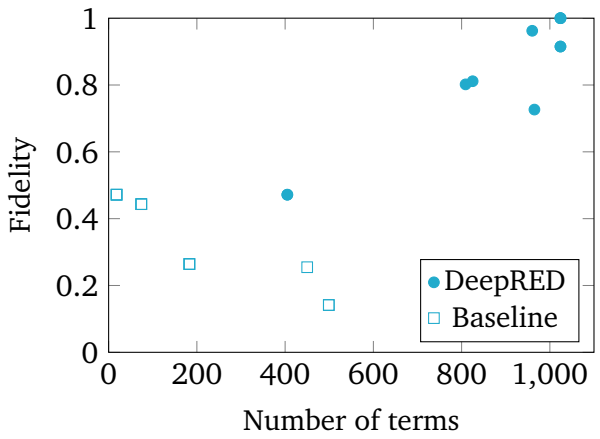


Figure 5.4: Evaluation results for DeepRED and the baseline on XOR.

Although we consider the MNIST problem a complex one, we won't go too deep into DeepRED's performance on this task, since the large number of inputs makes the task a difficult one for our algorithm. While the baseline, amongst other results, is able to extract rules with an overall number of eight terms that achieve a fidelity of about 94%[10], DeepRED offers two extremes: On the one hand it extracts a single rule with one single term that achieves a 87% fidelity, and on the other hand a 94% fidelity rule set with a total number of 120426 terms is extracted. The complex inner structure of the MNIST neural network does not seem to help achieving fairly comprehensible rules with acceptable fidelity rates – although a larger number of successful experiments would be necessary to further analyse DeepRED's problems with the MNIST neural network.

5.5.3 RxREN Pruning Helps DeepRED to Extract More Comprehensible Rules

In Section 4.1.2 we have already shown that RxREN pruning can facilitate the rule extraction process for DeepRED. In this section we want to analyse how the RxREN pruning affects the rule set's comprehensibility – and what it means for the rules' fidelity.

[10] This experiment's settings were 92% / ≤ 2% C4.5 / 10% data. Another experiment with the settings 99% / ≤ 0% / 50% led to a rule set with 271 terms that achieve a fidelity of approximately 96%.

Figure 5.5 gives some insights in how the different pruning settings (x-axis) affect the extracted rule set's comprehensibility (y-axis, logarithmic scale) while only those datasets are shown where not all inputs are important, i.e. letter, artif-I, and MNIST[11]. The points that are connected by a line represent experiments with the same C4.5 parameters and training set sizes. Please note, that the figure only shows groups of experiments where at least two different pruning settings were successful.

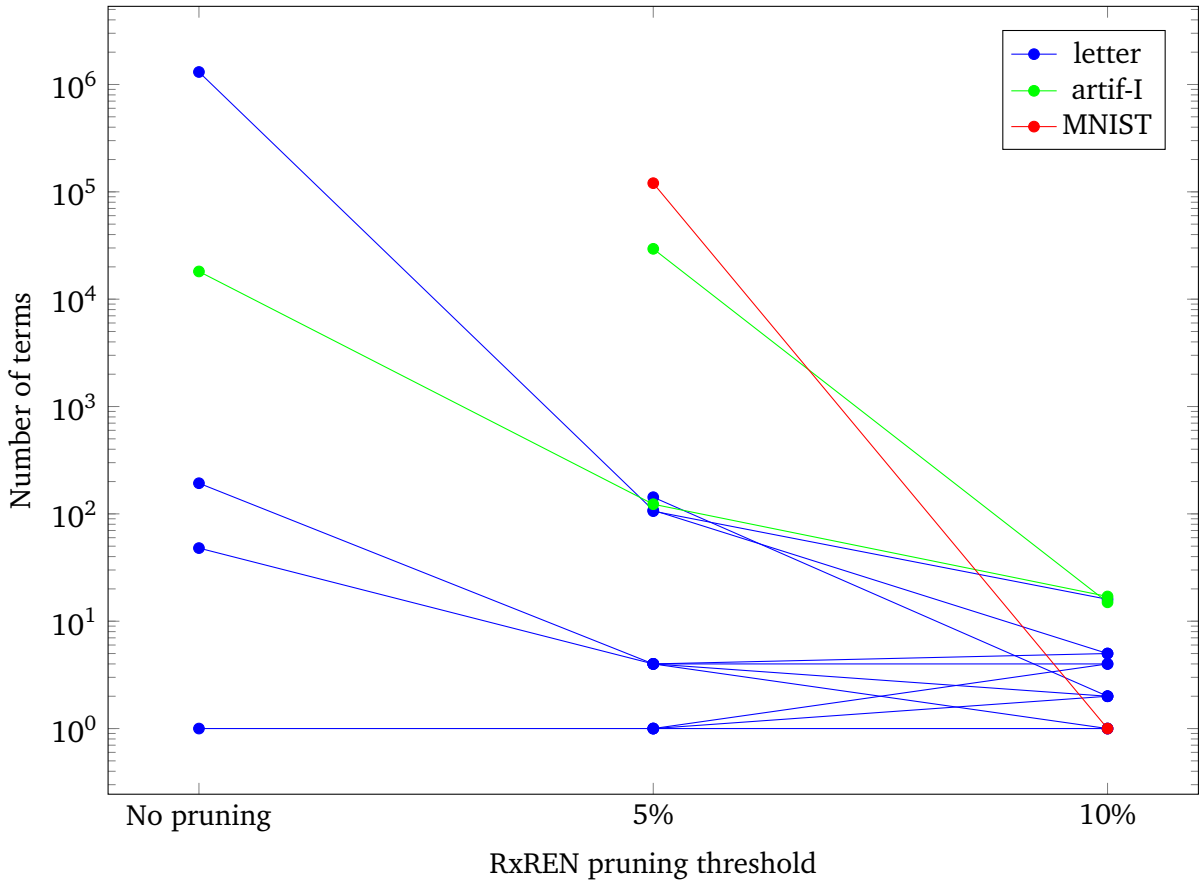


Figure 5.5: The effect of DeepRED's different RxREN pruning parameters on the extracted rules' comprehensibility.

One can clearly see that for the vast majority of experiments, the number of terms gets smaller when the pruning intensity gets higher. This holds true for all experiments when we only consider no pruning versus pruning with a threshold of 5%. For some runs of the letter problem, however, we can see slightly more comprehensible rules when applying RxREN pruning with a threshold of 10% instead of 5%. We argue that those changes can be ignored since the maximum increase is from one term to four terms which from our point of view does not affect the comprehensibility too much.

Pruning also affects the fidelity of the rules DeepRED extracts. Figure 5.6 provides fidelity rates for exactly the same experiments as depicted in Figure 5.5. For the artif-I problem, where the 10% pruning setting produced the shortest rules, we also see the best fidelity rates for the same pruning setting. For the MNIST problem, instead, the threshold of 10% is too greedy and leads to a much worse fidelity. For

[11] As mentioned earlier in Section 5.5, we consider experiments, where RxREN pruning should be applied but no pruning step could be performed given the threshold, as not executed. Therefore, showing the regarding artif-II and XOR results wouldn't lead to additional insights.

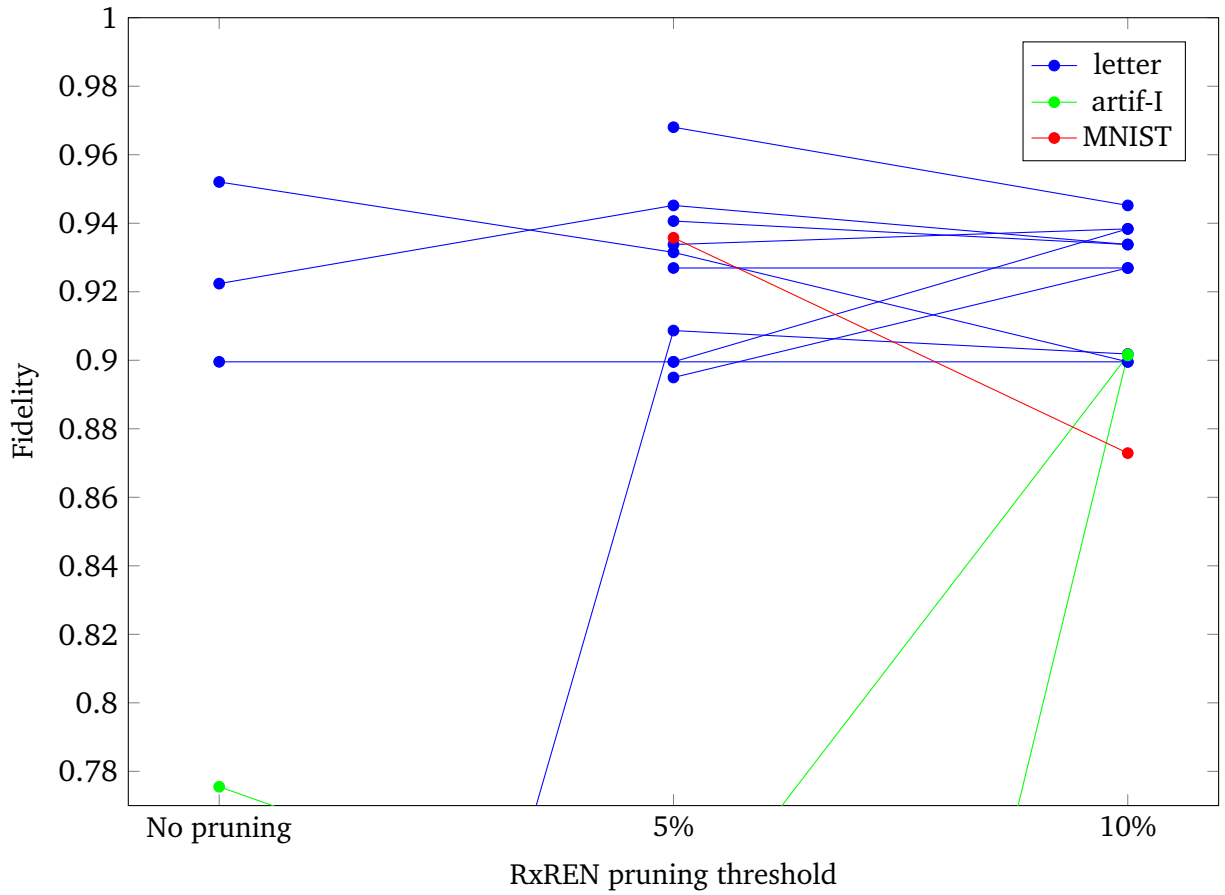


Figure 5.6: The effect of DeepRED's different RxREN pruning parameters on the extracted rules' fidelity.

future research, one could try a larger number of RxREN pruning settings that probably would lead to a good tradeoff setup.

The results for the letter problem are less clear. Again, we can see that the pruning intensity needs to be adjusted in a more elaborate way to ensure the best achievable results. But in total the results are better than expected since, although RxREN pruning overall increases the comprehensibility drastically, a clear trend towards lower fidelity rates cannot be recognized.

What the figures also show, is that RxREN pruning in eight of 13 cases in the first place enables DeepRED to successfully extract rules. Without the pruning option, for instance, DeepRED wouldn't be able to process the MNIST neural network at all.

5.5.4 DeepRED can Extract High-Quality Rules Independently from the Training Set Size

The last expectation we formulated in Section 5.4.1 refers to the training size that is available for the rule extraction algorithm. Our belief was, that DeepRED profits from more available data – although we expected a stronger correlation between the available data and the baseline's fidelity. Figure 5.7 provides us with the necessary information to analyse this matter. The graph shows the results of all successful experiments on datasets with less than 4000 training examples (with a minimum fidelity of 77%).

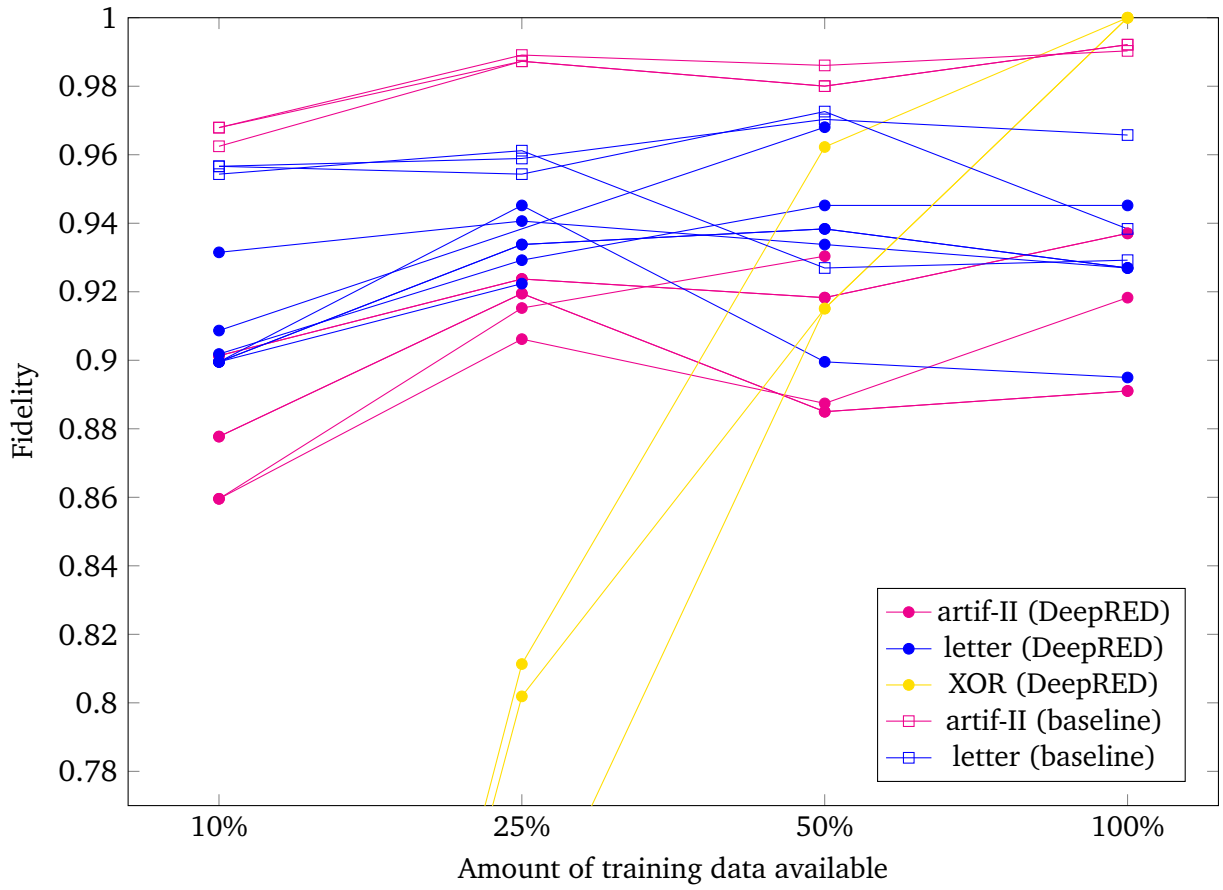


Figure 5.7: The implications of different sizes of the available data on the extracted rules' fidelity.

Although we can clearly see that increasing the training set size from 10% to 25% has a positive influence on the fidelity, the overall results for the artif-II and letter datasets are surprising. Contrary to our expectations more data don't necessarily lead DeepRED to extract better rules. Often, the fidelity even gets worse when making more data available to the proposed algorithm, while the baseline more often can profit from additional training data. In the future research, a further investigation, why this happens and how a variant of DeepRED could find the best training set size would be interesting.

For the XOR dataset, Figure 5.7 shows us exactly what we expected – a strong correlation between the number of training examples and the fidelity of the rule sets. While the results for the baseline aren't shown in the extract of the graph (since the results are never better than 50%), DeepRED manages to extract sensible rule sets with only 50% of the training data (which is about 29% of all possible input combinations). In our experiments with 100% training data, the proposed algorithm extracted rules that perfectly mimic the NN's behaviour. Again, we want to clarify that the 100% training data are approximately 59% of all possible input combinations. The baseline would only be able to extract such a rule set, if all 256 possible instances are available for training.

This clearly shows the benefit of having access to the neural network's structure. Considering the XOR neural network depicted in Figure 5.1, DeepRED needs four input combinations of I_1 and I_2 to find a rule for the first node in the second hidden layer. The same holds true for I_3/I_4 and the second neuron in this layer, and respectively for the other neurons in the second hidden layer. The advantage of DeepRED

is that it does not need every input combination of all attributes, but the examples only must contain the relevant input combinations of e.g. I_1 and I_2 anywhere in the training set. In the best case, for an XOR with four inputs, only four examples are needed[12], which is only 25% of all 16 possible input combinations. The deeper the network, the more DeepRED benefits from the NN's architecture.

5.5.5 Qualitative Analysis

In addition to the analysis of our expectations in the preceding sections, we also want to give a brief insight in how the rules extracted by DeepRED look like and how they relate to the original problems.

Earlier, we saw that DeepRED can outperform the baseline on artif-I (cf. Section 5.1), hence, we take a closer look at the extraction results for this problem. A rule set with a fidelity of approximately 93% on the test set, consisting of four rules and a total of twelve terms could be extracted by our algorithm. Although the original problem uses all five attributes, the extracted rules do not refer to x_4 or x_5 (due to RxREN pruning).

Two of the four extracted rules seem to realize the NN's representation of IF $x_1 = x_2$ THEN $\boldsymbol{y} = y_2$ (which is one of the original model's rules[13]): IF $x_1 \leq 0$ AND $x_2 > 0$ AND $x_2 \leq 0.5$ THEN $\hat{\boldsymbol{y}} = y_1$ as well as IF $x_1 \leq 0.5$ AND $x_2 > 0.5$ THEN $\hat{\boldsymbol{y}} = y_1$. We see that the extracted rules clearly prevent classifying an example as of class y_1 if the first two attributes are equal.

The other two rules seem to consider the original model's rule IF $x_1 > x_2$ AND $x_3 > 0.4$ THEN $\boldsymbol{y} = y_2$. The extracted rules are IF $x_1 > 0$ AND $x_1 \leq 0.5$ AND $x_2 \leq 0.25$ AND $x_3 \leq 0.38$ THEN $\hat{\boldsymbol{y}} = y_1$ and IF $x_1 > 0$ AND $x_2 \leq 0.75$ AND $x_3 \leq 0.38$ THEN $\hat{\boldsymbol{y}} = y_1$. Again we see, that the rules are reasonable and the threshold for x_3 is very close to the one of the original model.

On the other hand, we cannot identify an extracted rule that realizes the original model's third rule, i.e. IF $x_3 > x_4$ AND $x_4 > x_5$ AND $x_2 > 0$ THEN $\boldsymbol{y} = y_2$.

Analysing the results for arif-I, we can also provide evidence that DeepRED does not extract unnecessary rules. The rule set's fidelity decreases by at least 4 percent points if one of the rules is omitted. This is not the case for a comparable baseline result (approximately 92% fidelity, 35 rules with altogether 265 terms). Here several rules can be pruned without affecting the fidelity rates too much.

Although we want to clarify, that using the extracted rules as classifiers for unseen data is not the motivation behind rule extraction from neural networks, the analysis of this performance still might be interesting. We took a closer look at all extracted rule sets that contain less than 200 terms. Figure 5.8 shows the accuracy rates (y-axis) of the extracted rules on the test sets of the regarding problems (x-axis). The black points indicate the neural network's performance on these data.

We can clearly see, that the accuracy of the neural networks is better for all datasets, and that the extracted rules are not very close to the NN performances. Again, this underlines that rule extraction from neural networks should not be used to create rules for classification, but only to try to better understand a neural network's decision process.

[12] An example of a sufficient training set with the instance notation $\boldsymbol{x} = x_1x_2x_3x_4$ would be 0011, 1101, 1000, and 0110. It contains all combinations of x_1/x_2 and x_3/x_4.

[13] Please notice that this rule is relevant for classifying instances as class y_2 and not as class y_1.

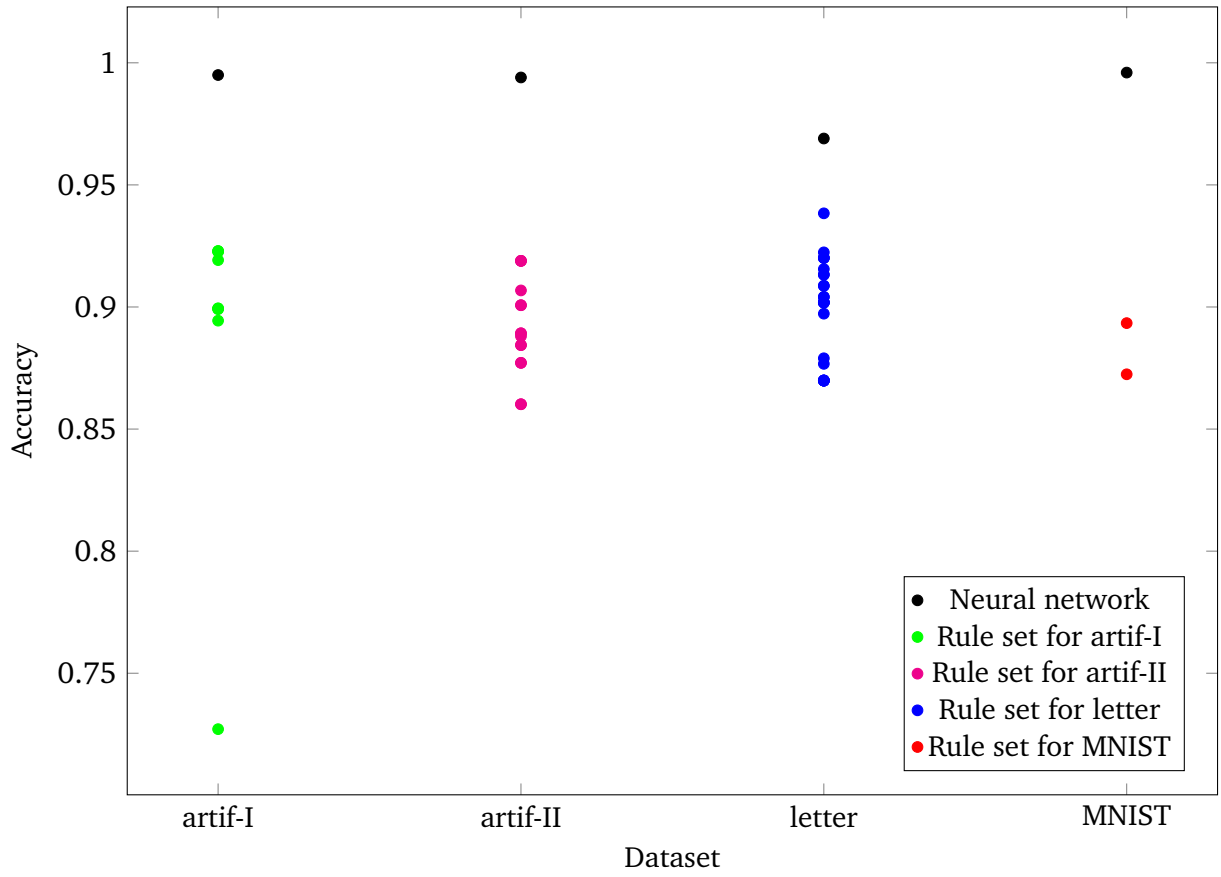


Figure 5.8: Comparison of the classification accuracy on the regarding test sets of extracted rule sets and the underlying neural networks.

Comparing the fidelity rates of the extracted rules, i.e. how well they mimic the neural network's behaviour, with the accuracy rates, i.e. how good they are in correctly classifying data to their true classes, reinforces DeepRED's focus. While for the artif-I, artif-II, and MNIST problems the extracted rules achieve accuracies very close to the fidelity rates, this is not the case for the letter problem. Here, the fidelity rates are approximately 3 percent points better than the regarding accuracy rates. In contrast to the letter problem, the other neural networks are very close to 100% accuracy while the letter NN achieves approximately 97%. Although we have only one dataset as evidence, this shows that the extracted rules really focus on the NN's behaviour instead of directly modelling patterns from the data.

5.6 Concluding Remarks

This chapter presented our evaluation of the DeepRED algorithm. After introducing the evaluation data as well as the rule extraction algorithms used in our experiments, we defined our experiment settings and discussed the results.

In Section 5.1 we introduced five classification datasets as well as the deep neural networks we built to solve the respective problems. The data include real-world problems and artificially created instances. We mainly focussed on four-layer neural networks, but also presented one NN with seven layers.

Afterwards, Seciton 5.2 summarized the parameters that can be changed to modify DeepRED's behaviour. This includes setting the threshold for the pruning mechanism, which has been adapted from the RxREN approach. We also presented the baseline – a simple pedagogical technique based on the C4.5 algorithm.

Section 5.4 provided the definitions of the central measures, i.e. the total number of terms present in the extracted rule set and the fidelity that measures how well the rules mimic a neural network's behaviour.

The final evaluation setup was introduced in Section 5.4. Here, we stated our expectations of how we thought DeepRED would perform the task of extracting rules from deep neural networks. We also presented the experiments meant to check our assumptions, before we provided the relevant information about the hardware setup.

The evaluation results have been provided by Section 5.5. We could observe that DeepRED in fact is able to successfully extract rules from all input data tested. We could even provide evidence, that the proposed algorithm can clearly outperform the baseline in two out of five cases. The results also showed that the implemented pruning mechanism helps DeepRED to extract better understandable rules. A surprising outcome was the fact that more training data does not necessarily help our approach to extract better rules. We finished Section 5.5 with a short qualitative analysis of the rules produced by DeepRED.

6 Conclusion

In this thesis, we analysed the problem of extracting rules from deep neural networks to make their decision processes more comprehensible. Besides an introduction to rule-based learning and neural networks, we gave an overview of the currently existing approaches and finally proposed and evaluated a new algorithm that can extract rules from DNNs.

Chapter 2 provided the foundations of machine learning that were necessary to understand the context of our work. Since this thesis focused on the classification problem, we introduced the general task and provided more detailed information on rule-based learning approaches and neural networks, and how they can be used to tackle these kinds of problems.

Chapter 3 presented the state-of-the-art in rule extraction research. We specified the problem and learned that there are three basic strategies to tackle them, i.e. decompositional, pedagogical, and eclectic approaches. From each of these groups, we introduced a number of available algorithms that can be used to extract rules from neural networks.

Since most state-of-the-art algorithms don't consider deep neural networks at all, we had to create a new approach. In Chapter 4, we proposed DeepRED – a new technique to extract rules from DNNs – which is based on one of the algorithms presented in Chapter 3. We also discussed the main challenges with deep neural networks and analysed DeepRED's way to handle them. Furthermore, in Chapter 4, we explained strategies to analyse single neurons in a DNN. These strategies are independent from the deep neural network rule extraction algorithm used.

In Chapter 5, we evaluated the proposed algorithm. We presented various datasets as a basis for our experiments and explained the parameters of DeepRED. At this point, we also introduced a simple baseline algorithm that is based on C4.5 and extracts rules pedagogically. The experiment setup as well as the evaluation results, have been presented in the same chapter. We found out that DeepRED works in general and that it can outperform the baseline, while pruning often helps. Surprisingly, the effect of more training data wasn't always positive.

The evaluation gave us some insights into the challenges and characteristics of DeepRED. We learned that the proposed algorithm does not work successfully with every arbitrary parameters. Also, there does not exist a single parameter setting that necessarily leads to good-quality rules. Therefore, a more extensive analysis of what settings are good for which environments, could be part of the future work on DeepRED. Of course, an automated fine-tuning of the pruning and C4.5 parameters would be even better. However, research has shown that finding clear parameter setting recommendations is not trivial.

We were surprised that DeepRED does not necessarily profit from larger training sets. The opposite is true: More available training examples can lead to worse rule sets. A first analysis could not answer the question why this is the case. Therefore, finding a more elaborate approach to select the training instances necessary to create the best results could be interesting, too. In future research, one could also analyse the effect of using sampled examples, just like the approaches presented in Section 3.3.2.

As introduced in Chapter 4, the current DeepRED implementation uses the C4.5 algorithm to create intermediate rules. However, DeepRED's approach is not dependant on C4.5. In future versions of our algorithm, intermediate rules could be created by any rule-based classification approach. This includes but is not limited to the RIPPER algorithm. For such variations, however, it must be kept in mind that rule lists would need to get transformed to rule sets. A further analysis of how rule-based regression learners can be applied to DeepRED could also be interesting.

Conclusively, the development of additional deep neural network rule extraction algorithms pursuing different strategies could lead to further sensible modifications and extensions of the DeepRED algorithm. Since, to the best of our knowledge, the DNN rule extraction problem by now has not been analysed more extensively by other researchers, we believe that there still is some room for improvement for our algorithm.

Bibliography

[Andrews et al., 1995] Andrews, R., Diederich, J., and Tickle, A. B. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389.

[Augasta and Kathirvalavakumar, 2012a] Augasta, M. G. and Kathirvalavakumar, T. (2012a). Reverse engineering the neural networks for rule extraction in classification problems. *Neural processing letters*, 35(2):131–150.

[Augasta and Kathirvalavakumar, 2012b] Augasta, M. G. and Kathirvalavakumar, T. (2012b). Rule extraction from neural networks - a comparative study. In *Pattern Recognition, Informatics and Medical Engineering (PRIME), 2012 International Conference on*, pages 404–408. IEEE.

[Bagallo and Haussler, 1990] Bagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine learning*, 5(1):71–99.

[Bengio et al., 2015] Bengio, Y., Goodfellow, I. J., and Courville, A. (2015). Deep learning. Book in preparation for MIT Press.

[Benítez et al., 1997] Benítez, J. M., Castro, J. L., and Requena, I. (1997). Are artificial neural networks black boxes? *Neural Networks, IEEE Transactions on*, 8(5):1156–1164.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer New York.

[Byson and Ho, 1969] Byson, A. E. and Ho, Y.-C. (1969). Applied optimal control. *Ginn & Co., Waltham, Toronto, London*.

[Chorowski and Zurada, 2011] Chorowski, J. and Zurada, J. M. (2011). Extracting rules from neural networks as decision diagrams. *Neural Networks, IEEE Transactions on*, 22(12):2435–2446.

[Ciresan et al., 2012] Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE.

[Cohen, 1995] Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, pages 115–123.

[Cortez and Embrechts, 2013] Cortez, P. and Embrechts, M. J. (2013). Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 225:1–17.

[Craven and Shavlik, 1999] Craven, M. and Shavlik, J. (1999). Rule extraction: Where do we go from here. *University of Wisconsin Machine Learning Research Group Working Paper*, pages 99–1.

[Craven and Shavlik, 1994] Craven, M. and Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In *ICML*, pages 37–45.

[Craven, 1996] Craven, M. W. (1996). *Extracting comprehensible models from trained neural networks*. PhD thesis, University of Wisconsin-Madison.

[Craven and Shavlik, 1993] Craven, M. W. and Shavlik, J. W. (1993). Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80.

[Craven and Shavlik, 1996] Craven, M. W. and Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, pages 24–30.

[Decloedt et al., 1996] Decloedt, L., Osório, F., and Amy, B. (1996). RULE_OUT method: A new approach for knowledge explicitation from trained ann. In *Proceedings of the Rule Extraction from Trained Artificial Neural Networks Workshop*, pages 34–42.

[Diederich et al., 2010] Diederich, J., Tickle, A. B., and Geva, S. (2010). Quo vadis? reliable and practical rule extraction from neural networks. In *Advances in Machine Learning I*, pages 479–490. Springer.

[Duch et al., 2004] Duch, W., Setiono, R., and Zurada, J. M. (2004). Computational intelligence methods for rule-based data understanding. *Proceedings of the IEEE*, 92(5):771–805.

[Freitas, 2014] Freitas, A. A. (2014). Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10.

[Frey and Slate, 1991] Frey, P. W. and Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6(2):161–182.

[Fu, 1994] Fu, L. (1994). Rule generation from neural networks. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(8):1114–1124.

[Fukumi and Akamatsu, 1998] Fukumi, M. and Akamatsu, N. (1998). Rule extraction from neural networks trained using evolutionary algorithms with deterministic mutation. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages 686–689. IEEE.

[Fürnkranz and Widmer, 1994] Fürnkranz, J. and Widmer, G. (1994). Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pages 70–77.

[Han et al., 2011] Han, J., Kamber, M., and Pei, J. (2011). *Data mining: concepts and techniques*. Elsevier.

[Haykin, 1994] Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan USA.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[Huysmans et al., 2006] Huysmans, J., Baesens, B., and Vanthienen, J. (2006). Using rule extraction to improve the comprehensibility of predictive models. *Available at SSRN 961358*.

[Huysmans et al., 2011] Huysmans, J., Dejaeger, K., Mues, C., Vanthienen, J., and Baesens, B. (2011). An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154.

[Jaitly et al., 2012] Jaitly, N., Nguyen, P., Senior, A., and Vanhoucke, V. (2012). Application of pretrained deep neural networks to large vocabulary speech recognition. In *Proceedings of Interspeech 2012*.

[Janssen, 2012] Janssen, F. (2012). *Heuristic Rule Learning*. PhD thesis, Technische Universität Darmstadt.

[Johansson et al., 2005] Johansson, U., König, R., and Niklasson, L. (2005). Automatically balancing accuracy and comprehensibility in predictive modeling. In *Information Fusion, 2005 8th International Conference on*, volume 2, pages 7–pp. IEEE.

[Johansson et al., 2006] Johansson, U., Lofstrom, T., Konig, R., Sonstrod, C., and Niklasson, L. (2006). Rule extraction from opaque models–a slightly different perspective. In *Machine Learning and Applications, 2006. ICMLA'06. 5th International Conference on*, pages 22–27. IEEE.

[Kane and Milgram, 1993] Kane, R. and Milgram, M. (1993). Extraction of semantic rules from trained multilayer neural networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1397–1401. IEEE.

[Kodratoff and Michalski, 2014] Kodratoff, Y. and Michalski, R. S. (2014). *Machine learning: an artificial intelligence approach*, volume 3. Morgan Kaufmann.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lichman, 2013] Lichman, M. (2013). UCI machine learning repository. http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.

[Loza Mencía, 2012] Loza Mencía, E. (2012). *Efficient Pairwise Multilabel Classification*. PhD thesis, Technische Universität Darmstadt.

[Montavon, 2013] Montavon, G. (2013). *On layer-wise representations in deep neural networks*. PhD thesis, Technische Universität Berlin.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[Otero and Freitas, 2015] Otero, F. E. and Freitas, A. A. (2015). Improving the interpretability of classification rules discovered by an ant colony algorithm: Extended results. *Evolutionary computation*.

[Palm, 2012] Palm, R. B. (2012). Prediction as a candidate for learning deep hierarchical models of data. Master's thesis, Technical University of Denmark.

[Pazzani, 2000] Pazzani, M. J. (2000). Knowledge discovery from data? *Intelligent systems and their applications, IEEE*, 15(2):10–12.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial intelligence: a modern approach*. Pearson Education.

[Saito and Nakano, 2002] Saito, K. and Nakano, R. (2002). Extracting regression rules from neural networks. *Neural networks*, 15(10):1279–1288.

[Sato and Tsukimoto, 2001] Sato, M. and Tsukimoto, H. (2001). Rule extraction from neural networks via decision tree induction. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1870–1875. IEEE.

[Schmitz et al., 1999] Schmitz, G. P., Aldrich, C., and Gouws, F. S. (1999). ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *Neural Networks, IEEE Transactions on*, 10(6):1392–1401.

[Seltzer et al., 2013] Seltzer, M. L., Yu, D., and Wang, Y. (2013). An investigation of deep neural networks for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7398–7402. IEEE.

[Sethi et al., 2012a] Sethi, K. K., Mishra, D. K., and Mishra, B. (2012a). Extended taxonomy of rule extraction techniques and assessment of kdruleex. *International Journal of Computer Applications*, 50(21).

[Sethi et al., 2012b] Sethi, K. K., Mishra, D. K., and Mishra, B. (2012b). KDRuleEx: A novel approach for enhancing user comprehensibility using rule extraction. In *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pages 55–60. IEEE.

[Setiono et al., 2008] Setiono, R., Baesens, B., and Mues, C. (2008). Recursive neural network rule extraction for data with mixed attributes. *Neural Networks, IEEE Transactions on*, 19(2):299–307.

[Setiono and Leow, 2000] Setiono, R. and Leow, W. K. (2000). FERNN: An algorithm for fast extraction of rules from neural networks. *Applied Intelligence*, 12(1-2):15–25.

[Setiono et al., 2002] Setiono, R., Leow, W. K., and Zurada, J. M. (2002). Extraction of rules from artificial neural networks for nonlinear regression. *Neural Networks, IEEE Transactions on*, 13(3):564–577.

[Setiono and Thong, 2004] Setiono, R. and Thong, J. Y. (2004). An approach to generate rules from neural networks for regression problems. *European Journal of Operational Research*, 155(1):239–250.

[Šíma, 1995] Šíma, J. (1995). Neural expert systems. *Neural networks*, 8(2):261–271.

[Szegedy et al., 2013] Szegedy, C., Toshev, A., and Erhan, D. (2013). Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561.

[Taha and Ghosh, 1999] Taha, I. A. and Ghosh, J. (1999). Symbolic interpretation of artificial neural networks. *Knowledge and Data Engineering, IEEE Transactions on*, 11(3):448–463.

[Thrun, 1993] Thrun, S. (1993). Extracting provably correct rules from artificial neural networks. Technical report, University of Bonn, Institut für Informatik III.

[Thrun, 1995] Thrun, S. (1995). Extracting rules from artificial neural networks with distributed representations. *Advances in neural information processing systems*, pages 505–512.

[Tickle et al., 1998] Tickle, A. B., Andrews, R., Golea, M., and Diederich, J. (1998). The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–1068.

[Towell and Shavlik, 1993] Towell, G. G. and Shavlik, J. W. (1993). Extracting refined rules from knowledge-based neural networks. *Machine learning*, 13(1):71–101.

[Tsukimoto, 2000] Tsukimoto, H. (2000). Extracting rules from trained neural networks. *Neural Networks, IEEE Transactions on*, 11(2):377–389.

[Vanthienen and Wets, 1994] Vanthienen, J. and Wets, G. (1994). From decision tables to expert system shells. *Data & Knowledge Engineering*, 13(3):265–282.

[Vellido et al., 1999] Vellido, A., Lisboa, P. J., and Vaughan, J. (1999). Neural networks in business: a survey of applications (1992–1998). *Expert Systems with applications*, 17(1):51–70.

[Zhou et al., 2000] Zhou, Z.-H., Chen, S.-F., and Chen, Z.-Q. (2000). A statistics based approach for extracting priority rules from trained neural networks. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 401–406. IEEE.

Thesis Statement

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. In the submitted thesis the written copies and the electronic version are identical in content.

Date/Signature:

Jan Ruben Zilke