Machine Learning Journal manuscript No.
(will be inserted by the editor)

DeepRED – Rule Extraction and Pruning in Deep Neural Networks

Camila González · Jan Ruben Zilke · Eneldo Loza Mencía

Received: date / Accepted: date

Abstract Neural network classifiers are known to be able to learn very accurate models. In the recent past, researchers have even been able to train neural networks with multiple hidden layers (deep neural networks) more effectively and 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 representations from neural networks. However, most authors focus on nets with only one hidden layer. The present paper introduces a new decompositional algorithm – *DeepRED* – that is able to extract rule models from deep neural networks. We analyze how the internal structure of the network influences the quality of the extracted rules, and introduce an algorithm to encourage a more desirable distribution of the logic. We also explore an approach that uses the network to easily determine what inputs are irrelevant, and excludes these from the rule extraction. Additionally, we show how applying a post-pruning technique after extracting the intermediate expressions and between merging steps can help control error propagation between layers while significantly reducing the size of the extracted models. The evaluation of the proposed algorithm shows its ability to outperform a pedagogical baseline on extracting symbolic representations that mimic more closely the behavior of the original network without significantly increasing the number of final terms.

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 (NN) are able to produce very low error rates that yet could not be achieved by other machine learning techniques (Johansson et al., 2006). Sufficiently deep NNs, so-called deep neural networks (DNN), are even 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.

Technische Universität Darmstadt
Knowledge Engineering Group
E-mail: camilag.bustillo@gmail.com, j.zilke@mail.de, eneldo@ke.tu-darmstadt.de

There is a major downside of NNs: For humans, it is not easy to understand how they derive their decisions (Russell and Norvig, 1995). Understanding a NN's function can be essential. For instance, this is the case in safety-critical application domains such as medicine, power systems, autonomous driving or financial markets, where a hidden malfunction could lead to life-threatening actions or enormous economic loss.

Also, in these domains machine learning models are not relied on by themselves but instead used to assist human actors in the form of decision support systems (DSS). Here, interpretability is of particular advantage, as the reasons behind a decision can make the human support or disregard it. This holds as well for anomaly detection systems, as for an alarm to result in an adequate response it should provide clear information describing why it was issued. Apart from the benefits of having insight into the inner working of the model, the extent to which the model is used in practice depends heavily on how easily interpretable it is, as this appears to be a relevant criteria for eliciting trust (Kayande et al., 2009; Gallego-Ortiz and Martel, 2016).

It is suggested by Lipton (2016) that a reason an incomprehensible model may not be trusted despite having a very low error rate is that the cost function may reflect partially, but not exactly, the goals of the user. For instance, a user may penalize a certain bias when manually taking decisions, but this may not be explicitly included in the objective the model optimizes. This is very common, as complex objectives are difficult to articulate in real-valued functions.

Ethical and juristic concerns are particularly warranted in recent times, given that the General Data Protection Regulation (GDPR)[1] is planned to take effect in 2018. The regulation outlines a set of practices for the handling of personal data which demand a significant adaptation of the way machine learning models are used (Goodman and Flaxman, 2016).

Within these, it is stated that some types of data should be excluded even if the way they were obtained adhere with the respective regulations, such as those that show the belonging of a subject to a group against which it is prohibited to discriminate. This also targets variables that are correlated with that information, even if they do not express it directly.

Perhaps because of the significant challenge this presents, as most variables are correlated to some degree, it is also stated that decisions which notably affect European citizens can only be made by automatic processing if it is under the supervision of a human, and that subjects have the right to receive explanations as to why they were taken. There is no outline as to what form such explanations should take beyond that they must be understandable.

In addition to the GDPR, there in an ongoing European legislative initiative on *Civil law rules on robotics* which also discusses "general and ethical principles governing the development of robotics and artificial intelligence for civil purposes".[2] The new legislation may raise liability issues for manufacturers, owners or users in case of robot's harmful behaviour. In the current first reading, it is also discussed whether manufacturers should provide "comprehensive design instructions including source code, similar to the legal deposit of publications to a national library", emphasizing the need for understandable descriptions of the functioning of artificial intelligence systems.

A better understanding of the way NNs derive their decisions could also push NN training research. Making NNs more transparent, for instance, could help to discover so-called hidden features that might be formed in DNNs while learning. Such features are not present

[1] Regulation (EU) 2016/679, http://www.eugdpr.org/

[2] Under the ID of 2015/2103(INL): http://www.europarl.europa.eu/oeil/popups/ficheprocedure.do?reference=2015/2103(INL)

in the plain input data, but emerge from combining them in a useful way. Such hidden information, inherent in especially deep networks, was recently referred to as *dark knowledge*.

In contrast to NNs, rule-based approaches like decision trees or simple IF-THEN rules are known to be better understandable. Since most humans tackle classification problems in a manner very similar to the one implemented by many rule-based learning techniques, i.e., listing simple conditions that need to be met, their models and decision processes are more comprehensible.

To overcome the weakness of NNs being black boxes, especially in the 1990s researchers have worked on the idea of extracting rules from them. 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 NNs with only one hidden layer. Surprisingly little work has been done on analyzing the challenges of extracting rules from the more complex DNNs. 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 DNNs. But indeed, having a large amount of layers makes the extraction of comprehensible rules more difficult.

There is no consensus regarding what makes a model comprehensible. Lipton (2016) divides interpretability into two characteristics. The first is transparency, or insight into the model as a whole, which depends among other factors on the size of the model, the speed to which a human can build it manually and the extend to which he or she can conceptually understand the parts that make it up. The second covers all additional information that may be extracted from the model, such as insight into relevant input or hidden features. This information may be global or with respect to one decision, which could for instance take the form of very similar observations in the train set.

Therefore, it would seem that the goal of interpretability can either be pursued at model level or by providing explanations for individual decisions. Model explanation systems take, for instance, the second path by illustrating the output of individual classifications (Turner, 2016), thus providing justification without sacrificing the predictive accuracy of the model as a whole.

In this paper, we introduce and evaluate *DeepRED*, a new algorithm that is able to extract rules from DNNs. The decompositional algorithm extracts intermediate rules for each layer of a DNN. A final merging step produces rules that describe the DNN's behaviour by means of its inputs.

We show how different simplification and post-pruning techniques employed after extracting the intermediate expressions and between merging steps can help to assure that the complexity of the final rules remains similar to that of pedagogical approaches that neglect the inner layers. We also explore how applying two different pre-pruning mechanisms to DNNs can assist the latter rule extraction while not significantly compromising the networks' predictive capacity.

The rules extracted with *DeepRED* can serve as an additional model which leverages the features learned by deep neural networks. Alternatively, the algorithm can be used to explain why classifications are made for each class, and even to justify single predictions by extracting rules describing the value of each output neuron.

The evaluation shows *DeepRED*'s ability to successfully extract rules from DNNs that are highly faithful to the original network while remaining interpretable in a way that matches or outperforms a pedagogical approach. For datasets that model some complex concepts, logical rules extracted with *DeepRED* are also more accurate than those derived directly from the data.

The work is structured as follows: First, we give an overview of the current state-of-the-art. Afterwards, we describe our proposed algorithm in more detail (Section 3) and introduce network and rule pruning techniques in Section 4. We evaluate *DeepRED* on several tasks and under different considerations (Section 5 and 6) before we summarize and conclude our work.

2 Related Work

The following section presents relevant work specific to the extraction of rules from neural network and the pruning of networks in order to leverage the extraction process. Furthermore, we revise related research on model compression and induction of interpretable models.

2.1 Extraction of rules from neural networks

It is commonly believed that "concepts learned by neural networks are difficult to understand because they are represented using large assemblages of real-valued parameters" (Craven and Shavlik, 1994). To transform NNs into a form that is better comprehensible for humans, many concepts are conceivable. One famous choice is to extract rules from NNs. Andrews et al. (1995) have introduced a widely used taxonomy to distinguish between different rule extraction algorithms. The taxonomy, for instance, comprises the *translucency* dimension that defines the strategy used: decompositional (considering NN's inner structure), pedagogical (black box approach), or eclectic (mixture of both).

Examples of eclectic methods are the *MofN* algorithm (Towell and Shavlik, 1993) as well as the *FERNN* algorithm (Setiono and Leow, 2000). *MofN*'s main approach is to find NN connections with similar weights. *FERNN* focusses on identifying a NN's relevant inputs and hidden neurons.

The works dealing with pedagogical rule extraction, for instance, comprise the *VIA* method discussed by Thrun (1993, 1995), different sampling-based approaches (Craven and Shavlik, 1996; Taha and Ghosh, 1999; Schmitz et al., 1999; Zhou et al., 2000; Sethi et al., 2012), and the *RxREN* algorithm (Augasta and Kathirvalavakumar, 2012). *VIA* uses validity interval analysis to find provably correct rules. Sampling-based methods try to create extensive artificial training sets where rule learning algorithms later can extract rules from.

Examples for decompositional approaches are the *KT* method that heuristically searches for input combinations that let a neuron fire (Fu, 1994), a more efficient implementation of the latter (Tsukimoto, 2000), and an algorithm that transforms NNs to fuzzy rules (Benítez et al., 1997). However, the most important basis for this present work is the (decompositional) *CRED* algorithm presented by Sato and Tsukimoto (2001). *CRED* uses decision trees to describe a NN's behaviour based on the units in its hidden layer. In a second step it builds up new decision trees to describe the split points of the first decision trees. Afterwards rules are extracted and merged.

Although we encounter a wide variety of interesting rule extraction approaches, there is a major shortcoming: Most decompositional algorithms introduced so far cannot deal with NNs with more than one hidden layer. Furthermore, to the best of our knowledge, there does not exist any algorithm that has explicitly been tested on the task of extracting rules from DNNs. This is the case even though most pedagogical approaches should be able to perform this task without any major modifications. However, due to the lack of pedagogical

techniques to include the knowledge present in a DNN's inner structure, we focus on a decompositional method to extract rules from DNNs.

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 by Cortez and Embrechts (2013) (cf. also Section 2.4). 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 support vector machines Diederich et al. (2010).

2.2 Pruning of Neural Networks

Pruning connections of trained neural networks is a common practice, as it is a way to adapt the topology of the network to the effective size of the problem, thus discouraging overfitting and increasing the generalization capabilities. It can also be leveraged to, among other benefits, require fewer training examples and improving the speed of classifying new ones (LeCun et al., 1989; Thodberg, 1991; Hassibi et al., 1993; Anwar et al., 2015).

A connection between two neurons can be pruned by equaling the weight entry that connects them to zero. It differs from applying dropout in that connections are removed permanently, and which connections to prune is decided according to some criterion, whereas dropout consists on temporarily removing randomly chosen connections for a training step to reduce overfitting.

Connection pruning is usually carried out in the following manner:

1. Networks are trained using back-propagation until a satisfactory loss is reached. The cost function may or may not include a regularization term.
2. One or more weights are pruned, chosen by some heuristic as being the least significant.
3. The network is retrained until an acceptable accuracy has been reached or a certain number of training steps have been performed. If the accuracy is acceptable, pruning is resumed; otherwise the last acceptable solution is chosen.

Thimm and Fiesler (1997) compare five different heuristics to select the next weight to be pruned in multilayer perceptrons. The results show that the pruned networks often, but not always, achieved a better generalization, and the simplest heuristics, which chose the weights with lowest magnitude, often performed best.

Han et al. (2015) and Tang and Han (2015) apply connection pruning to networks with fully connected and convolutional layers that were trained on image datasets. Weight decay is performed during back-propagation before pruning weights with magnitude below a threshold. They found that retraining after each pruning step allows to reduce the number of connections by between nine and thirteen times without loss in accuracy.

In (Setiono, 1997b), a similar method is presented to prune connections from fully connected feed-forward networks with one hidden layer. This approach is used by Setiono and Liu (1996); Setiono (1996) to extract rules from neural networks. After the networks have been trained and pruned, the hidden unit activations are discretized using clustering. Then, rules $R_{0\to1}$ and $R_{1\to2}$ are build greedily using the discretized activation, and $R_{0\to2}$ is found by applying substitution and resolving conflicts. Rules created by this method had on average more conditions but less rules than rule sets found using a tree building approach build with the same dataset, and the predictive accuracy of the network was at large maintained.

Setiono (1997a) extend the approach by introducing a new hidden layer into the network if the number of inputs to a hidden unit exceeds some threshold. After retraining, the extracted rule sets showed to have higher accuracy and less rules.

In (Kamruzzaman and Hasan, 2010), the algorithm *REANN* first grows the number of hidden units in a shallow NN until an acceptable accuracy is reached. Irrelevant connections are removed by weight elimination and the activations are discretized before extracting the rule representation. *RxREN* (Augasta and Kathirvalavakumar, 2012) focus only on the weakest input features and is described in more detail in Section 4.

In contrast to the methods described above which focus on the weakest connections, the approach developed in this work (Section 4.2) additionally tries to balance the number of connections used between any pair of layers. The motivation is avoiding a high concentration of logic at some of the layers and hence impulsing the exploitation of the deep architecture.

2.3 Model compression and knowledge distillation

The idea of *model compression* is to train a compact model to imitate the behaviour of another, more complex system, such as a large ensemble of classifiers (Bucila et al., 2006). Despite being simple, it is often observed that the mimicking student model is able to reach performance levels of the more complex teacher system and at the same time being much more efficient. Note that it is assumed that applying the student learner to the original data would lead to inferior performance, hence the usage of a more powerful teacher model. In the context of neural network training, where this general approach is often referred to as *knowledge distillation*, a shallow neural network is often used in order to predict the actual outcome, i.e., the numeric outputs at the last layer, of a more complex NN. Targeting the actual activations is considered to be essential for knowledge distillation to work.[3] It is assumed that the advantage over directly applying the shallow net to the original data mainly comes from the additional information which the teaching system provides by expressing uncertainties or confidences, not only about the true class or the class it believes is true, but also about every other class. For instance, Hinton et al. (2015) was able to show that it is possible to predict a class which was never seen during training just by feeding the mimicking learner with probability estimates for that class on the training instances. This implicit information contained in the outputs was later referred to as *dark knowledge*. In a sense, we can consider knowledge distillation as a special case of *learning using privileged information* (Vapnik and Vashist, 2009), i.e., the setting where additional information is available about the training data which will however not be present during the predicting phase.

Ba and Caruana (2014) introduces the idea to use shallow neural nets to mimic deeper nets. They found that shallow nets are able to approximate the deeper teacher nets with a high accuracy whereas training the shallow net on the original net would only achieve low accuracies. They conclude that using deep neural nets overcome the limitations of current training methods which hinder to train smaller nets effectively on datasets where they could, in principle, perform well. These findings served as a motivation for the development of our proposed approach. Mimicking deep NN layerwise could enable obtaining symbolical decision models for classification problems for which current training methods such as classical decision tree learning and separate and conquer rule induction are not suitable. In contrast to

[3] Note that applying a weak learner on the discretized predictions of an almost perfect classifier would result in an almost equal model as training the weak learner directly on the original dataset.

common knowledge distillation approaches, our proposed approach tries to access the dark knowledge at the nodes and layers rather than only at the outputs. We believe that such more fine grained mimicking method could help to make the knowledge visible which is inside the network and to better identify the specific source of high accuracy.

On the other hand, a follow-up study shows that mimicking approaches have its limitations (Urban et al., 2017). Complex deep convolutional architectures form an obstacle which cannot be cleared by simple NNs anymore. Instead, the authors have to use a few layers of CNNs in order to reach performance levels of the original deep nets. Hence, we exclude CNN from the scope of this work, since we suspect that the complex structure and logic of CNN can hardly be reflected by simple symbolical languages.

2.4 Comprehensive models and predictions

There is an increased interest in interpretable and transparent machine learning system in the recent past. Only in 2016 three workshops took place, namely the Workshops on Human Interpretability in Machine Learning(kim, 2016), on Interpretable Machine Learning for Complex Systems (wil, 2016), and on Fairness, Accountability, and Transparency in Machine Learning[4]. Hence, the following section should just introduce some examples techniques in order to give a broad overview over the field.

Apart from symbolical models, logistic regression models are considered to be well comprehensible by humans. A logistic regression model is a linear combination of the (numeric) input variables weighted each by a coefficient. Hence, it allows to inspect which features are relevant and what the contribution was to a prediction. However, for problems with a high input dimensionality, such systems lose interpretability due to the high number of weights. Using L1-regularization such as in LASSO (Tibshirani, 1996) may alleviate the problem since it results in many coefficients being zero. Zeng et al. (2016) present Supersparse Linear Integer Models (SLIM), which is basically a logistic regression learner which learns integer coefficients for binary input features, and only for as little input variables as possible. SLIMs allows to present the prediction model as a scoring system, where the presence of characteristics sums up points until a threshold is reached and the test example is predicted as positive.

Caruana et al. (2015) demonstrated the benefit of using Generalized Additive Models (GAM) over less powerful approaches, such as linear logistic regression, and over less intelligible approaches, such as random forests, on a case study on predicting and visualizing pneumonia risk. GAM are similar to logistic regression in that each input feature contributes additively to the target value (e.g., risk). However, in GAM the contribution does not depend linearly on the feature value, but on a non-linear function. The model can be visualized by graphs of these non-linear functions. The plots allow to identify intervals of higher risk (for instance age, respiration rate) and the concrete contribution.

If more complex learners are desired or needed, direct visualization attempts quickly face limitations. (Kasneci and Gottron, 2016) follow a decompositional approach in order to quantify the influence of input variables in neural networks by approaching locally the behaviour of individual nodes by piecewise linear function. In contrast, the method LIME is model-agnostic, i.e., just as the pedagogical approach it treats the classifier to mimic as a black-box Ribeiro et al. (2016). In addition, it does not aim at producing a global interpretable model, but at providing an comprehensive explanation for a particular prediction

[4] http://www.fatml.org/

of a classification system. To this end, it selects training instances in the locality of the test instance at hand and trains an interpretable classifier such as LASSO on these, which is then presented to the user.

The latter approach was also used to explain text classification with non-linear SVMs. However, most attempts on understanding complex systems can be found in the computer vision domain, where (deep) neural networks have become state-of-the-art in the recent past. Attempts in image recognition using neural networks usually try to identify the regions or individual pixels which were responsible for a particular classification. Bach et al. (2015) use layer-wise relevance propagation in order to create heat-maps for visualization. The technique used by Zeiler and Fergus (2014) just systematically covers an image with a moving gray block in order to plot a heat map of picture regions whose occlusion would change the prediction.

Image recognition systems also bring up the issue of trust and safety. For instance, Sharif et al. (2016) demonstrate that it is quite easy to deceive state-of-the-art neural networks used for face recognition: by wearing glasses with colorful patterns printed on it it was possible to appear some other target person with a high success rate. Unfortunately, domains which are safety-critical such as medical robots, self-driving cars, etc. are exactly those where machine learning systems are the least trustworthy, as Varshney and Alemzadeh (2016) point out. The reason are the many sources of uncertainties in the models due to the shortage of especially critical observations.

Hence, we consider it to be essential that complex systems in critical domains can be inspected beforehand by experts and believe that *DeepRED* can provide the means for that in many scenarios. In addition, as it relies on a symbolical representation for its decisions, the decisive input variables, and even decisive connections between input variables, for a particular prediction become obvious very naturally. As we will see in the following section, rules are selected according to their coverage of training instances. These associations to specific examples could be used in order to support a user in estimating the trust in a particular decision.

3 The DeepRED Algorithm

With *DeepRED* (Deep neural network Rule Extraction via Decision tree induction), in this paper we present an algorithm that is able to overcome the shortcoming of most state-of-the-art approaches by being applicable to DNNs. Since *DeepRED* is a decompositional method, in contrast to pedagogical ones, the algorithm is able to easily extract hidden features.

This section is structured as follows: First, we introduce our notation and assumptions. After describing the *CRED* algorithm in more detail, we present *DeepRED*'s way to extract rules from DNNs as well as the pruning technique used.

3.1 Preliminaries

We assume a multiclass classification problem with m training examples $x_1, \ldots, x_m$, each x_j associated with one class $y_j \in \{\lambda_1, ..., \lambda_u\}$. We represent the input values of a NN as $i = i_1, \ldots, i_n$ and the output values as $o = o_1, \ldots, o_u$, one for each possible class. The hidden layers are abbreviated as $h_i \in \{h_1, \ldots, h_k\}$ while the hidden layer h_i consists of the neurons $h_{i,1} \ldots, h_{i,H_i}$ (in the case of a single-hidden-layer NN, the hidden neurons are written as $h_1, \ldots, h_H$). For convenience, we set $h_0 = i$ and $h_{k+1} = o$ and let $h_i(x)$ denote the specific

layer values for input instance x. *DeepRED* produces intermediate rule sets $R_{a \to b}$ that include rules that describe layer b by means of terms based on layer a. A rule r : IF *body* THEN *head* is constituted by a set of terms in the body (conditions on attribute values, or, in our case, also conditions on activation values) and an assignment term in the head.

3.2 CRED as Basis

As mentioned earlier, *DeepRED* is based on *CRED* (Continuous/discrete Rule Extractor via Decision tree induction). In a first step, the original algorithm introduced by Sato and Tsukimoto (2001) uses the well-known *C4.5* algorithm Quinlan (1993) to transform each output unit of a NN (with one hidden layer) into a decision tree. Such a tree's inner nodes are tests on the values of the hidden layer's units. The leaves represent the class such an example would belong to (cf. Figure 1). Afterwards, intermediate rules are extracted directly from these trees. In the case illustrated in Figure 1, 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 $\hat{y} = \lambda_1$. This leads us to a rule set that describes the behaviour of a NN on the basis of the hidden layer's units.

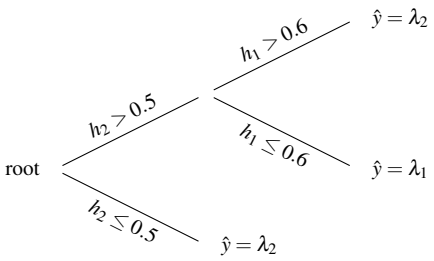

Fig. 1 Result of CRED's first step: A decision tree providing rules for the neural network's output based on its hidden layer (adapted from Sato and Tsukimoto (2001)).

Next, for each term used in these rules, another decision tree is created using split points on the NN's input layer. In these new decision trees, the leaves do not directly decide for an example's class, but rather on the tests used in the first tree's node, as exemplarily depicted in Figure 2. 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 by means of the hidden layer and those that describe the hidden layer based on the NN's inputs are substituted and merged to build rules that describe the NN's output on the basis of its inputs.

3.3 Extending CRED to DeepRED

Since *CRED* is a decompositional rule extraction approach, it is not possible to apply the original implementation directly to NNs with multiple layers of hidden neurons. However,

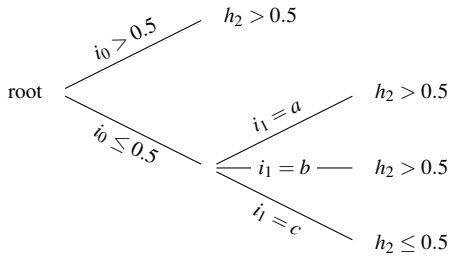

Fig. 2 Result of one iteration of CRED's second step: A decision tree providing rules for $h_2 > 0.5$ based on the neural network's inputs (adapted from Sato and Tsukimoto (2001)).

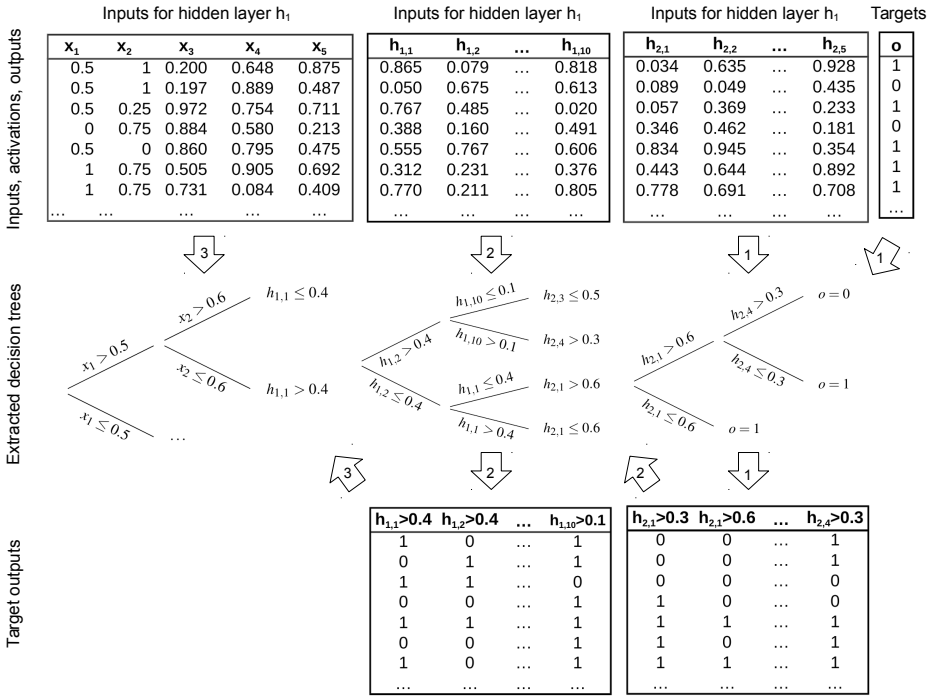

Fig. 3 Illustration of the way *DeepRED* builds rules from the target condition through the intermediate layers by building decision trees with the activation values of the train data.

the algorithmic approach can be extended in a relatively straight-forward manner by deriving additional decision trees and intermediate rules for every supplementary hidden layer.

The general process of *DeepRED* is the following: The algorithm extracts rules according to the state of one output neuron, such as that its activation is higher than of all other output neurons (for classification problems) or within an activation range (for regression problems). It processes every hidden layer in a descending order and extracts rules for each layer that describes its behaviour based on the preceding layer. In the end, all rules for the target concept are merged such that we arrive at the rule set $R_{i \to o}$. An illustration of the process is provided in Figure 3.

Input: Neural network $h_0, h_1, \ldots, h_k, h_{k+1}$, training examples $x_1, x_2, \ldots, x_m$
Output: Set of rules representing the NN

1 **foreach** *class* $\lambda_v \in \lambda_1, \ldots, \lambda_u$ **do**
2 $R^v_{h_k \to o} \leftarrow$ IF $h_{k+1,v} > 0.5$ THEN $\hat{y} = \lambda_v$ `// initial rule for NN's prediction`
3 **foreach** *hidden layer* $j = k, k-1, \ldots 1$ **do**
4 $R^v_{h_{j-1} \to h_j} \leftarrow \emptyset$
5 $T \leftarrow$ `extractTermsFromRuleBodies` $(R^v_{h_j \to h_{j+1}})$
6 $T \leftarrow$ `removeDuplicateTerms` (T)
7 **foreach** $t \in T$ **do**
8 $x'_1, \ldots, x'_m \leftarrow h_j(x_1), \ldots, h_j(x_m)$
`// obtain activation values for all neurons in the` $j-1$`-th layer for each training example`
9 $y'_1, \ldots, y'_m \leftarrow t(h_{j+1}(x_1)), \ldots, t(h_{j+1}(x_m))$
`// apply term` t `on respective activation in layer` j `and get binary outcome, the training signal, for each training example`
10 $R^v_{h_{j-1} \to h_j} \leftarrow R^v_{h_{j-1} \to h_j} \cup$ `deleteRedundantTerms(C4.5` $((x'_1, y'_1), \ldots, (x'_m, y'_m))$
`// learn rules for` t `and add to rules representing the layer`
`// merging to conjunctive rules and cleaning up`
11 **foreach** *hidden layer* $j = k, k-1, \ldots 1$ **do**
12 $R^v_{h_{j-1} \to o}$ = `mergeIntermediateTerms(` $R^v_{h_{j-1} \to h_j}$, $R^v_{h_j \to o}$ `)`
`// dissolve rules of remaining two layers`
13 $R^v_{h_{j-1} \to o}$ = `deleteUnsatisfiableTerms(`$R^v_{h_{j-1} \to o}$`)`
14 $R^v_{h_{j-1} \to o}$ = `deleteRedundantTerms(`$R^v_{h_{j-1} \to o}$`)`
15 return $R^1_{i \to o}, R^2_{i \to o}, \ldots, R^u_{i \to o}$ `// describes output of NN by rules consisting of terms on input values of first layer`

Fig. 4 Pseudo code of *DeepRED*.

Our modified version of the *CRED* algorithm 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. Figure 4 provides the pseudo code for *DeepRED*. To exemplify *DeepRED*'s approach we consider, without loss of generality, a NN with k hidden layers. As a result of the first step we get a decision tree that describes the output layer by split points on values regarding h_k, i.e., the 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 (line 8).

In the next step, in contrast to the algorithm presented by Sato and Tsukimoto, we do not directly refer to the input layer, but instead process the next shallower hidden layer, i.e. h_{k-1} (loop starting in line 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 $R_{h_{k-1} \to h_k}$ (line 10).

Just like in the *CRED* example presented earlier, the terms in $R_{h_k \to o}$ are directly used to differentiate positive and negative examples for the regarding *C4.5* runs (line 9). We see that the proposed algorithm implements a procedure to prevent itself from performing redundant runs of *C4.5* since we only learn a decision tree for terms which were not already extracted (line 6). Also, the intermediate expressions are simplified to prevent building decision trees for redundant terms (line 10).

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 NN, i.e. $R_{i \to h_1}$.

Now we have rule sets that describe each layer of the NN 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 (block starting at line 11).

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 $R_{h_{k-1}\to o}$ (line 12). As mentioned in Figure 4, unsatisfiable intermediate rules as well as redundant terms are deleted (lines 13 and 14), which happens to reduce computation time and memory usage drastically. This is outlined in more detail in Section 3.5.

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 NN, which is the result we were looking for.

3.4 Extraction of DNF formulas from trees

Within *DeepRED*, each tree determines whether an example exceeds an activation value for a certain neuron. For instance, a tree could be built to determine if the value 0.5 is exceeded by the second neuron in the first hidden layer. Such a tree would have two possible classes, $h_{1,2} > 0.5$ and $h_{1,2} \leq 0.5$. The tests within the tree are conditions of the same type, but with respect to neurons from the next shallower layer. Each leaf that predicts a class determines a rule for that class. Rules are extracted from the tree by joining with conjunctions all conditions that delimit a path from the root condition to a leaf, which is labeled with a class. This generates rules of the form

$$X_1 \wedge X_2 \wedge \ \ldots \ \wedge X_n \to class$$

with $X_1, X_2, \ldots, X_n$ as the conditions in the inner nodes from the root to one of the leafs of the tree. A DNF formula is built for a class by joining all these rules with a disjunction. A DNF formula for the event of neuron $h_{1,2}$ exceeding the threshold 0.5 may look like:

$$(h_{0,1} > 0.5 \wedge h_{0,2} \leq 0.3) \vee (h_{0,2} > 0.3 \wedge h_{0,3} > 0.7) \ \to h_{1,2} > 0.5$$

If both opposite class conditions are used in rules of the next deeper layer, only one tree is built and both DNF representations are extracted and stored. For instance, if both $h_{1,2} > 0.5$ and $h_{1,2} \leq 0.5$, which refer to the second neuron of the first hidden layer, acted as terms for rules describing the second hidden layer, then one tree would be built and two different DNFs would be extracted.

However, as *DeepRED* only extracts rules for one output target class at a time, all rules predict at any time only for one class and the ordering does not affect the prediction of the rule set. Therefore, rules can be altered even if this means the two DNF representations keep being disjunct, as they are right after being removed from the tree.

3.5 Elimination of redundant and unsatisfiable terms

Once a DNF representation is created for the target condition, and another is created for each condition included in it, replacement takes place so that the formula of the output conditions goes from being in terms of the last hidden layer to using conditions of the previous layer. This process is repeated until the conditions in the expression are in terms of the inputs.

This can be very time and memory demanding, and it is possible that rules are created which are logically inconsistent, or strictly weaker to other rules in the set. To keep the number of terms at a reasonable number, redundant and inconsistent rules and terms are deleted

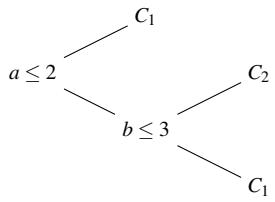

Fig. 5 A decision tree that may result in an equivalent DNF representation with redundant terms.

each time a replacement step is carried out. *DeepRED* removes rules that are unsatisfiable, rules that are redundant within the DNF and conditions which are more general than others in the same rule.

A rule is *unsatisfiable* if it includes two conditions that are mutually, exclusive, such as the following:

$$h_{0,2} > 0.3 \wedge h_{0,2} \leq 0.2 \ \rightarrow \ h_{1,2} > 0.5$$

A rule is *redundant* if it is strictly weaker than another rule in the same rule set. Rule r_1 is strictly weaker than rule r_2 if it includes terms for all dimensions delimited by r_2, and either restricts on further dimensions or the restrictions are at least as specific than those of r_2. In the following expression, the second and third rules are redundant:

$$h_{0,2} > 0.3 \vee (h_{0,2} > 0.3 \wedge h_{0,4} \leq 0.2) \vee h_{0,2} > 0.4 \ \rightarrow \ h_{1,2} > 0.5$$

A term is also redundant if the rule includes a more specific term. That is the case for the first term of the following example rule:

$$h_{0,2} > 0.3 \wedge h_{0,4} \leq 0.2 \wedge h_{0,2} > 0.2 \ \rightarrow \ h_{1,2} > 0.5$$

3.5.1 Additional detection of redundant rules and terms

A common pitfall affects DNF representations extracted from decision trees, which is that the sequential manner in which the tree is evaluated results in rules sometimes having irrelevant conditions. For instance, consider the decision tree in Figure 5. The expression extracted from this tree to classify C_1 would be $a \leq 2 \ \vee (a > 2 \ \wedge \ b > 3) \rightarrow C_1$, although it is clear that the first condition on the second conjunction is irrelevant, as if the test $a > 2$ was to fail, the instance would also belong to class C_1.

This addition of unnecessary conditions is outlined by Quinlan (1987a,b). He proposes an algorithm for converting decision trees into small sets of production rules of the form *IF <conditions> THEN <class> (<certainty factor>)*, where every condition that does not pass a probabilistic significance test is removed. The resulting rule sets are not equivalent to the original trees but a generalization of the model. In the experiments performed these rules achieve better accuracy than the trees.

Preliminary experiments were performed that applied this type of pruning to the DNF representations extracted within *DeepRED*. However, although the complexity of intermediate expressions was strongly reduced, the pruning proved to be too aggressive and considerably affected the fidelity.

Still, this pitfall can be addressed in a lossless manner without having to consider the training data by expanding the syntactic deletion of redundant terms. The following two cases are considered:

– Two rules that classify for the same class are identical except that the second has a condition such that, if not fulfilled, the first rule would fire (plus perhaps additional conditions). In this case, the irrelevant condition is deleted from the second rule. For example, $(a > 2 \wedge b \leq 3 \wedge c > 2) \vee (a > 2 \wedge b <= 3 \wedge c \leq 6 \wedge d > 5) \rightarrow C_1$ would be transformed to $(a > 2 \wedge b \leq 3 \wedge c > 2) \vee (a > 2 \wedge b \leq 3 \wedge d > 5) \rightarrow C_1$

– Two rules that classify for the same class are identical except for one condition in each conjunction such that if one was not fulfilled, the other would be. For instance: $(a > 2 \wedge b \leq 3 \wedge c > 3) \vee (a > 2 \wedge b \leq 3 \wedge c \leq 5) \rightarrow C_1$. In this case, the complemented condition is deleted and the two rules are merged into one, as $a > 2 \wedge b \leq 3 \rightarrow C_1$.

The second case is not likely to be generated from a decision tree, but often occurs during merging when conditions are replaced by expressions of a shallower layer.

4 Pruning of network connections and extracted rules

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 (cf. Section 2).

As a pre-processing step before *DeepRED* is applied, we have borrowed the input pruning technique from *RxREN* (Augasta and Kathirvalavakumar, 2012). It proceeds by testing the NN's performance while ignoring individual inputs. Those inputs that where not necessary to still produce an acceptable classification performance are getting pruned. The method is described in more detail in the following subsection.

Instead of pruning complete inputs, the *weight sparsity pruning* technique introduced in Section 4.2 considers to prune irrelevant connections between layers aiming at obtaining a sparsely connected network.

We expect the network pruning techniques to enable *DeepRED* to extract more compact rules without losing too much accuracy. However, as was already anticipated in Section 3.5, the conversion of decision trees to rules and the merging of intermediate rules leaves us with a potentially high number of rules, which are likely also overlapping. In addition to the lossless filtering techniques already presented, we introduce a post-pruning technique which maintains a high fidelity in Section 4.3.

4.1 RxREN pruning

In their work, Augasta and Kathirvalavakumar (2012) introduce a method that prunes the neural network first before it is used for rule extraction. Their proposed algorithm RxREN (rule extraction by reverse engineering the neural network) focuses on generating rules that are as comprehensible as possible.

The neural network pruning is implemented by analyzing the network's performance while ignoring individual input attributes (i.e., they are set to zero). If the NN's behaviour does not alter significantly without the regarding attribute, the input and all the associated connections are pruned. 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 inputs to solve a classification task, while at the same time reducing the extracted rules' potential complexity.

As RxREN performs the pruning independently of the rule extraction process, this technique is also well suited for being used as a pre-processing step in our proposed system.

More specifically, before *DeepRED* mimics a NN, our implementation of RxREN iterates over all inputs and evaluates the accuracy drop on the train set when the input is pruned. Subsequently, the inputs with the lowest decreases are removed until the a predetermined maximum accuracy difference is reached. On networks that use a great number of unprocessed input features, this approach allows a high percentage of them to be pruned while only causing a small decrease in accuracy. For instance, from our network trained using the MNIST data set (cf. Section 5.1.2), 21.3% of the inputs were pruned when allowing for a maximum 5% accuracy decrease, and the actual decrease in accuracy was only 2.3%.

4.2 Weight Sparseness Pruning

We developed a network pruning technique similar to the connection pruning techniques described in Section 2.2. In contrast to those methods, ours does not aim to reduce the total number of connections but to sparsify the weight matrices so that the total number of connections between any two layers is reduced. Pruning is thus not concentrated on weight matrices of high dimensions, or matrices where the magnitude of the weights are generally lower than in other matrices, but spread among all weight matrices in the same manner. This has the effect that single neurons are neither connected to a majority of the neurons of the following layer nor to a majority of those from the previous layer. The expectation is that, as observed by Setiono (1997a), rules extracted from minimally connected neurons are simpler and more accurate.

The motivation for targeting such connections also comes from the performance of *DeepRED* when applied to a network manually constructed to emulate the XOR function with eight inputs, shown in Section 6. The network constructed for this experiment has a recursive structure from the eight inputs to the output layer and is minimally connected (each neuron has a maximum of two incoming and two outgoing connections). *DeepRED* is not only able to extract the modeled DNF representation using a significantly lower number of instances than the pedagogical approach, but its intermediate rules also exactly replicate the recursive features.

This effect is not repeated on fully-connected networks of the same topology trained with back-propagation, even if all combinations are used for training the DNN. When rules are extracted from such networks using a reduced subset of combinations, *DeepRED* extracts simpler intermediate rules that overfit to the currently available examples, and concentrates the majority of the logic on rules between the inputs and the first hidden layer. These rules vary greatly depending on the train instances used, and each DNF extracted for layer h_1 depends on the majority of the inputs.

If, on the other hand, the number of connections was reduced, the network may be encouraged to learn a reduced amount of hidden features that are more abstract and apply to a greater percentage of examples. By forcing each neuron to only use a subset of the neurons of the previous layer, the logic should be more distributed.

4.2.1 Basic methodology

The proposed method is described in more detail in the following. The pseudo code is depicted in Figure 6.

A connection $W_i^{r,c}$ is represented by the tuple (i,r,c), where i is the index of the weight matrix, r the row and c, the column. The number of entries that have already been pruned in each row or column of each weight matrix is kept track of to calculate the *neighborhood sparseness* of the remaining entries. This value is determined by, for an entry (i,r,c), the sum of entries that have been pruned in row r of matrix W_i plus those that have been pruned in column c of the same matrix.

At the start, a list of all existing connections is constructed (line 3). On each step, the list is sorted in terms of decreasing neighborhood sparseness (line 7) and it is attempted to prune the next head element (the least sparse, which is likely surrounded by unpruned entries).

The target train accuracy that must be reached after retraining for the eliminated connection to remain pruned is equal to the original train accuracy minus an allowed accuracy decrease. If the accuracy is satisfactory, the connection is pruned, the counts for column and row pruned entries are updated and the list is re-sorted. Otherwise the last accepted parameters are restored and the next least sparse connection is removed from the list.

4.2.2 Iterations used for retraining

Preliminary experiments showed that it becomes clear during a small number of iterations what connection will not be able to be pruned, as there is no change in the accuracy and the network gets stuck in a local minimum. However, some connections cause a steep decrease in accuracy when they are first removed, but the network is able to adapt after a considerable number of iterations.

To allow the latter connections to be eliminated while not considerable increasing the retraining time, it was decided to take a different approach, which is to subdivide the total iterations for retraining into smaller sets. The procedure keeps track of two different parameters: *no improvements* and *steps*, which are initialized to zero each time a new connection is chosen.

Instead of performing all retraining steps at once, a low number of training steps (around 1% of those needed to train the network) are performed, after which the accuracy is calculated. If it is equal or greater than the target accuracy, the connection is pruned; otherwise that value is stored as the *last accuracy* achieved and another set of iterations is carried out. Afterwards, the accuracy is recalculated. If it is smaller or equal than the *last accuracy*, the count for *non-improvements* is increased by one, otherwise it is reset to zero and the new accuracy is adopted. The number of *retraining steps* is increased regardless of the value of the accuracy. The retraining phase is stopped when either parameter reaches its predefined maximum.

To make sure the process is not stretched out because of the accuracy cycling between a small set of values, the *last accuracy* is always updated to the maximum of the new accuracy and the initial accuracy after eliminating the connection (line 25).

4.2.3 Re-exploration of a connection

We observed that if a connection had not been able to be pruned once, it was extremely unlikely that it would be able to be pruned later on, even if other connections affecting the neurons it joined had been pruned. Also, if these connections were to remain as yet to explore, the sorting function would make the process choose them each time a connection had been pruned before still unseen connections, considerably slowing down the training. Therefore, it was decided that it would only be attempted to delete each connection once. If the target accuracy was not reached, the connection would no longer be considered.

```
   Input  : network, trainData, maxNonImprovements, maxSteps, iterations, accuracyDecrease
   Output: pruned network
 1 targetAccuracy ← network.getAccuracy (trainData)*(1-accuracyDecrease)        // use separate
   validation set if enough data available
 2 parameters ← network.getBiasWeights
 3 possibleIndices, prunedInRow, prunedInColumn ← initPossibleIndices (network.arch ())
 4 prunedIndices ← {}
 5 while possibleIndices ≠ {} do
 6     nonFound ← True
 7     possibleIndices ← sparcitySort (possibleIndices, prunedInRow, prunedInColumn)
 8     forall the (i,r,c) ∈ possibleIndices do
 9         possibleIndices ← possibleIndices \ {(i,r,c)}
10         network.pruneConnection ((i,r,c))
11         lastAccuracy ← network.getAccuracy (trainData)
12         nonImprovements ← 0
13         runs ← 0
14         while nonImprovements < maxNonImprovements & steps < maxSteps & nonFound do
15             for k ∈ [0, iterations) do
16                 network.train (trainData)
17             steps ← steps + 1
18             accuracy ← network.getAccuracy (trainData)
19             if accuracy>=targetAccuracy then
                                /* the decrease in accuracy is acceptable, hence prune */
20                 prunedIndices ← prunedIndices ∪ {(i,r,c)}
21                 prunedInRow_{i,r} ← prunedInRow_{i,r} + 1
22                 prunedInColumn[i][c] ← prunedInColumn + 1
23                 parameters ← network.getBiasWeights
24                 nonFound ← False
25             else if accuracy > lastAccuracy then
26                 lastAccuracy ← max (accuracy, lastAccuracy)
27                 nonImprovements ← 0
28             else
29                 nonImprovements ← nonImprovements + 1
30         if nonFound then
31             network.setBiasWeights (parameters)
32         else
33             break                    // break from forall so connections are reordered
34 return network
```

Fig. 6 Pseudo code of weight sparcity pruning method.

4.3 High fidelity post-pruning

The previous models addressed the reduction of the complexity of the final rule model by pruning a neural network before the extraction process is started. During the extraction process, however, the problem remains that after each replacing step while building the final expression, we may still obtain a great number of rules, many of which are not syntactically inconsistent or redundant but are extremely similar to other rules in the set. Besides affecting the comprehensibility, often the elevated number of rules propagates the error of the intermediate decision trees, which results on the final expression having a high recall but low precision on the test data.

To address this problem, a method for reduced error pruning method was implemented that only makes a change on the rule set if this does not affect, or positively affects, the

fidelity towards the target condition. Briefly, the proposed approach works as follows. A more detailed representation is provided by the pseudo-code in Figure 7.

For each rule, the change in overall fidelity is calculated for each of the following cases:

- The rule is deleted
- A condition is removed from the rule (checked for all conditions of the rule)
- The rule is merged with another rule from the set (checked for all remaining rules in the set)

Ideally, the repercussions of carrying out all possible modifications for every rule in the rule set would be calculated on each step, and the best would be carried out. However, this was found to require too long in cases where the number of rules created after a replacement step was very high. Therefore, it was decided that the rules would be handled one after the other, choosing the best modification for a rule before considering the next one.

Rules are ordered in terms of their increasing precision $P(r) = \frac{true\ positives}{true\ positives + false\ positives}$ (line 3) so that unprecise rules are handled first. A heap structure stores each yet unseen rule, so that new rules can be efficiently introduced. Each time a rule is observed, all modifications that can be made from it, as well as their fidelity, are calculated. Calculating the varying fidelity scores is sped up by putting apart all examples that fire by any other of the unseen rules. A list is also maintained for each remaining rule of the example indices it covered.

For each rule, the modification which leads to the highest fidelity is chosen, and if the fidelity is higher or equal than when leaving the rule unchanged, the original rule is removed from the rule set. Unless the modification consists on removing the rule completely, the certainty factor is calculated for the new rule and it is inserted into the heap of unseen rules. This goes on until there are no unseen rules. Of course, the approach could allow some decrease in fidelity, but this was not explored in the context of this work as it would have required an additional hyper-parameter tuning. Instead, the last computed fidelity is successively increased or adopted unchanged.

It was explored in preliminary experiments whether it would be advantageous to exclude a subset of the available training data from building the trees and instead reserving it to perform a less biased post-pruning. However, it was consistently found that the fidelity was higher when all the available data was used to build the decision trees, and the same data was used for post-pruning.

4.3.1 Merge between two rules

The *CRED* algorithm includes a step which simplifies intermediate rule sets by merging rules into fewer more general ones that are each more *interesting*. Merging two rules means selecting the most general condition of the dimensions they share, and dropping all conditions of dimensions they do not share, as exemplified below:

$$\begin{aligned} \text{Rule 1:}\quad & a \leq 0.3 \wedge b > 4 \wedge c > 2 \rightarrow C_1 \\ \text{Rule 2:}\quad & a \leq 0.2 \wedge b > 3 \wedge d > 2 \rightarrow C_1 \\ \text{Merge of rules 1 and 2:}\quad & a \leq 0.3 \wedge b > 3 \rightarrow C_1 \end{aligned}$$

The interestingness measure used is $J(X,Y) = p(X)p(Y|X)log_2\frac{p(Y|X)}{p(Y)}$, where X refers to the rule and Y to the class value. A rule is thus more interesting than its predecessors if it fires more often than the original rules while maintaining a high correlation between firing and the class condition being fulfilled.

```
Input  : single DNF, pruning set X, targetCondition
Output: pruned DNF
1  remainingRules ← DNF
2  lastFidelity ← computeFidelity(V, DNF, targetCondition)
3  unseenRules ← heap({(precision(r), r) | r ∈ DNF })
4  while unseenRules ≠ {} do
5      rule ← pop (unseenRules)
6      ruleVariants ← { emptyRule }
7      ruleVariants ← ruleVariants ∪ { merge (rule, other) | other ∈ remainingRules \ {rule} }
8      ruleVariants ← ruleVariants ∪ { rule \ {condition} | condition ∈ rule }
         /* The order in which ruleVariants is incremented causes that in the event of a
       tie in the certainty factor, deleting the rule is chosen first, merging second
       and removing a condition last. */
9      foreach r ∈ ruleVariants do
10         fidelity_r ←computeFidelity(X, remainingRules \ {rule} ∪ {r}, targetCondition))
11     newRule ← argmax_{r∈ruleVariants} fidelity_r
12     maxFidelity ← max_{r∈ruleVariants} fidelity_r
13     if maxFidelity ≥ lastFidelity then
14         lastFidelity ← maxFidelity
15         remainingRules ← remainingRules \ {rule}
16         if newRule ≠ emptyRule then
17             unseenRules ← unseenRules ∪ {(precision(newRule), newRule)}
18             remainingRules ← remainingRules ∪ { newRule}
```

Fig. 7 Pseudo code for High Fidelity Post-pruning

Applying this approach was tested in preliminary experiments and it was found that, although it is successful in reducing the complexity of the rules, when used to extract rules from deep neural networks it has a strong negative effect on the fidelity of the final expression. This is so because a rule that is much more likely to occur leads to a gain in interestingness even if it is less accurate. It also very often occurs that rules are merged that only share a small subset of the original dimensions, having a too strong generalizing effect.

However, if only merges are performed which increase or have no affect on the fidelity, the benefits of reducing the number of rules and terms can be leveraged without considerably affecting the accuracy. We therefore included performing a merging with another rule from the rule set as an alternative transformation within high fidelity post-pruning, outlined in Section 4.3.

5 Experimental Setup

Since we are not aware of any algorithm that has been tested on extracting rules from DNNs, we want to fill this gap with a *DeepRED* evaluation. In our view, the two most important aspects concerning the extraction of rules from NN are the obtained fidelity to the original network on the one hand and the model complexity on the other hand (cf. Section 5.3). The performance of *DeepRED*, the proposed improvements, and other relevant parameters of the extraction process should mainly be analyzed under these two aspects. Section 5.2 provides an overview over the algorithms and algorithm variants we evaluated together with the possible parameters we evaluated and compared to.

Although *DeepRED* is able to extract rules from any arbitrary activation range of an output neuron, in this study we limit ourselves to classification problems, for which *DeepRED*

Table 1 Overview of datasets and neural networks used for evaluation. Part 1. The first six datasets are examined in the first phase of the evaluation, and the remaining ones in the second.

dataset	# NN inputs	# NN outputs	# training examples	# test examples
Artificial 1	5	2	20000	10000
Artificial 2	5	2	3348	1652
Monks 1	6	2	302	130
Monks 3	6	2	303	129
Balance Scale	4	2	375	
Wilt	5	2	2904	1935
MNIST class '1'	784	10	12056	2195
Letter Recognition class 'A'	16	26	1239	438
Breast Cancer	30	2	341	228
Banknote Authentification	4	2	823	549
Page Blocks	10	2	3284	2189
Monks 2	6	2	259	173

disregards the output value itself and instead considers as the target condition whether the network output respective of the selected class exceeds those of all other outputs.

Before we present and discuss the results more extensively, we first take a look at the evaluation data base (Section 5.1).

5.1 Data Sets and Neural Networks for Evaluation

To evaluate *DeepRED*, we need suitable input data, i.e. trained NNs and training sets. Unfortunately, it is not trivial to find already trained NNs in literature or online. Although research very often addresses particularly the training of NN, usually only network structures, training methods and performances are reported, while specific weights are not of particular interest for most researchers.[5]

Therefore, we trained DNNs for real-world and artificial problems. The topology we chose for the DNNs consists of three hidden layers. For a network with n_0 inputs and n_4 outputs, the first hidden layer has $n_1 = min(30, 2n_0)$ neurons, the second has $\lfloor \frac{n_1+n_4}{2} \rfloor$ and the third has the same width as the output. The networks use the hyperbolic tangent activation function and a softmax function in the final layer. An exception are the networks for the XOR problem, which use a sigmoid activation and have a different topology. Tables 1, 2 and 3 summarize our data basis and the performance and structure of the NNs. Note that the accuracy rates reported for the *MNIST* and *Letter Recognition* data sets are based on the data sets reduced to two classes. Also note that the NNs for boolean datasets were trained with 100% of the data and have therefore a perfect accuracy.

5.1.1 Artificial datasets 1 and 2

The datasets *artificial 1* and *artitifical 2* were created by us and comprise examples with five attributes (discrete and continuous domains). A challenging characteristic of *artificial 1* is that it contains greater-than relations between real-valued attributes. These functions cannot easily be modeled by decision trees. This is not the case with *artificial 2*. However,

[5] Excluding the recent effort of making pre-computed image recognition networks publicly available, for instance by Simonyan and Zisserman (2014). For the reasons already outlined in Section 2.3, such networks are out of scope for this work.

Table 2 Overview of datasets and neural networks used for evaluation. Part 2, showing the structures and accuracies of the neural networks.

dataset	NN structure	NN train accuracy	NN test accuracy
Artificial 1	5-10-6-2-2	1.0	0.9983
Artificial 2	5-10-6-2-2	1.0	1.0
Monks 1	6-12-7-2-2	1.0	1.0
Monks 3	6-12-7-2-2	1.0	1.0
Balance Scale	4-8-5-2-2	1.0	0.992
Wilt	5-10-6-2-2	0.989	0.9933
MNIST class '1'	784-30-20-10-10	0.9907	0.9897
Letter Recognition class 'A'	16-30-28-26-26	0.9733	0.9657
Breast Cancer	30-30-16-2-2	1.0	0.9693
Banknote Authentification	4-8-5-2-2	1.0	1.0
Page Blocks	10-20-11-2-2	0.9896	0.9896
Monks 2	6-12-7-2-2	1.0	0.8786

Table 3 Boolean functions and topologies of the networks trained to model them.

	Function	# inputs	# examples	Network topology
XOR	$A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H$	8	256	8-8-4-4-2-2-2
CNF 1	$(\overline{F} \vee C \vee E) \wedge (A \vee D \vee F) \wedge (B \vee D)$	6	64	6-12-7-2-2
CNF 2	$(\overline{D} \vee E \vee F) \wedge (\overline{G} \vee \overline{H} \vee C) \wedge (A \vee D \vee F) \wedge (B \vee D)$	8	256	8-16-9-2-2
DNF 1	$(\overline{F} \wedge C \wedge E) \vee (A \wedge D \wedge F) \vee (B \wedge D)$	6	64	6-12-7-2-2
DNF 2	$(\overline{D} \wedge E \wedge F) \vee (\overline{G} \wedge \overline{H} \wedge C) \vee (A \wedge D \wedge F) \vee (B \wedge D)$	8	256	8-16-9-2-2

the second artificial dataset is based on a model where the value of one attribute has no effect on the class, which as well might be challenging for rule extraction algorithms.

More specifically, input instances were drawn randomly from $x \in \{0, 0.5, 1\} \times \{0, 0.25, 0.5, 0.75, 1\} \times [0,1]^3$ for both data sets. For *artificial 1* the class label $y = \lambda_1$ is generated by applying the rule "IF $(x_1 = x_2) \vee (x_1 > x_2 \wedge x_3 > 0.4) \vee (x_3 > x_4 \wedge x_4 > x_5 \wedge x_2 > 0)$", otherwise we set $y = \lambda_2$. The dataset *artificial 2* follows the rule "IF $x_1 = x_2 \vee (x_1 > x_2 \wedge x_3 > 0.4) \vee x_5 > 0.8$ THEN $y = \lambda_1$ ELSE $y = \lambda_2$".

5.1.2 Data sets from the UCI Machine Learning Repository

The *MNIST* database as found in the UCI repository (Lichman, 2015) describes handwritten digits from zero to nine by a number of 784 greyscale attributes, i.e. values from zero to 255. The basis for our *MNIST* data set is the one presented by LeCun et al. (1998). We trained a network with the original train data, but for rule extraction we picked a subset of the original data and reduced the problem to distinguish only between two different classes (*1* vs. rest).

The *Letter Recognition* problem is derived from the dataset introduced by Frey and Slate (1991). The letters are also represented with pixels, but these have been transformed by the authors to a set of 16 attributes. Here, as well, we first trained the DNN on a multi-class training set and afterwards reduced the training and test set to a binary classification problem.

The *Wisconsin Diagnostic Breast Cancer* data set contains various features describing digitalized images of breast mass cell nuclei. Each example consists of thirty real-valued input features and the dataset contains 357 instances classified as *benign* and 212 as *malignant*. We extracted rules for the first class.

The *Wilt* data set identifies diseased trees in Quickbird images and was originally collected in a study by Johnson et al. (2013). The features are segments from the pansharpened

image. The data set has a rather uneven class distribution, as 261 instances belong to the *diseased* class and 4578 to the *not diseased* one. Rules were extracted for the former class.

The *Balance Scale* data set classifies if a balance scale is being tipped to the right, tipped to the left or balanced. The attribute values are the weights and distances for the left and right sides. We transformed this dataset into a two-class problem before training the network, as there are only 49 instances for the *balance* class and we wanted to ensure a satisfactory predictive accuracy towards that class of the NN without taking special care of the training process. The transformed set contains 288 instances for the *right* class and 337 for the other two classes.

The examples for the *Banknote Authentification* dataset are made out of four attributes describing the image of each banknote and a classification pointing to the note's authenticity. The attribute values are the variance, skewness, curtosis and entropy of the wavelength transformed image. There are a total of 1372 images, of which 684 are for the first class, for which rules were extracted, and 762 for the second.

The *Page Blocks* data set classifies weather a page block is a text, horizontal or vertical line, graphic or picture. Each example is represented by ten input features extracted from page block images. Within those are the height, length and area of the block, as well information regarding the grayscale values of the pixels. Again, we converted the data into a two-class problem before training the network, as all classes other than *text* were very under-represented. The set is made out of 4913 instances for the *text* class, for which rules were extracted, and 560 for the other classes.

The *Monk's Problems* datasets were used for the first international comparison of learning algorithms Thrun et al. (1991). These consist of three artificial datasets which model, respectively, the following logical patterns:

- Monk's 1: $(a1 = a2) \vee (a5 = 1)$
- Monk's 2: Two of $\{a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 1\}$
- Monk's 3: Two of $(a5 = 3 \wedge a4 = 1) \vee (a5 \neq 4 \wedge a2 \neq 3)$

The original data includes a train and test set for each problem, with the test set containing all combinations of possible input values $a3, a6 \in \{1,2\}$, $a1, a2, a5, a6 \in \{1,2,3\}$, $a5 \in \{1,2,3,4\}$, which make a total of 432, and the train set only a subset of these. As no other dataset used contained examples from the test set in the train data, we made a new division of the test instances for each problem. The third Monk's problem had originally a 5% class noise added to the train set. This was excluded from our newly partitioned Monk's three data set.

5.1.3 Boolean datasets

The parity function is well-known as a hard problem for rule learners. Top-down learners, for instance, need all possible input examples to correctly model *XOR*. For this problem, only those instances belong to the first class where the number of attributes, that are set to one, is odd. This results in a database with a maximum number of 256 distinct instances.

To test whether *DeepRED* could extract rules from neural networks that modeled such a problem, we applied it to a manually constructed deep neural network with a total number of seven layers, as well as to a fully-connected automatically trained one with the same architecture. For both networks, the first output neuron is meant to fire if an example belongs to the first class.

The manually constructed DNN is depicted in Figure 8. The weights are shown at the regarding connections, the numbers in the neurons are the respective biases. It uses several

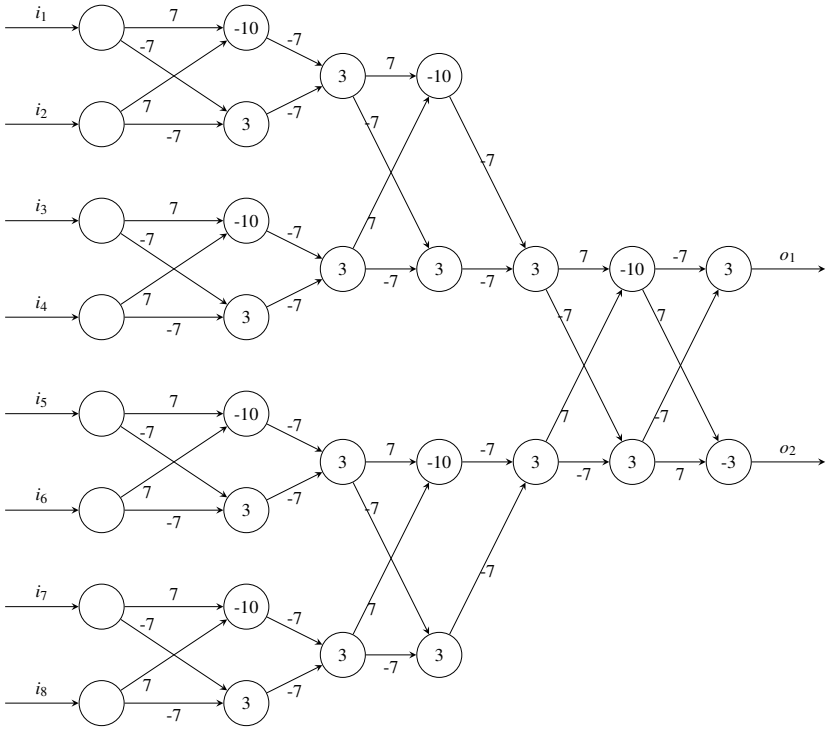

Fig. 8 A deep neural network solving the XOR problem with eight attributes

XOR functions on two inputs to recursively create a 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 $\overline{(i_1 \wedge i_2)} \wedge (\overline{\bar{i_1} \wedge \bar{i_2}})$. The DNN trained with gradient descent was trained using the entirety of the data so that it also had a perfect accuracy.

In addition to the XOR problem, four two-level expressions with six and eight inputs were randomly generated. The difference between both expression is their minimal form, which is either in disjunctive or conjunctive normal form. Again, the networks were trained and pruned using all combinations so they achieve perfect accuracy. The boolean datasets and the topology of the networks are shown in Table 3.

5.2 Algorithms and Parameters

To obtain an idea of which modifications to *DeepRED* help extract the most faithful and comprehensible rules, we compare a total of six *DeepRED* configurations between each other and with the corresponding pedagogical alternatives. A variant is defined by whether Weight Sparseness Pruning, RxREN pruning or no network pruning was performed, as well as by whether the intermediate and final rules were subjected to High Fidelity Post-pruning.

We perform experiments using different percentages of the data to inspect how the approaches fare when different amounts of data are available. Preliminary experiments have

Table 4 Overview of the effect of applying RxREN pruning and Weight Sparseness Pruning (WSP) in the neural networks used for the evaluation. It shows the number of inputs, the percentage of connections pruned, and the resulting accuracies on the train and test sets after the pruning achieved by the respective approaches.

dataset	# inputs pruned with RxREN	% of connections pruned with WSP	RxREN train	RxREN test	WSP train	WSP test
Artificial 1	0	83.33	1.0	0.9983	0.99	0.9899
Artificial 2	1	84.92	0.9943	0.9921	0.9910	0.9867
Monks 1	3	91.38	1.0	1.0	1.0	1.0
Monks 3	2	91.95	1.0	1.0	1.0	1.0
Balance Scale	0	75.58	1.0	0.992	0.993	0.98
Wilt	1	89.68	0.9724	0.9752	0.9824	0.9865
MNIST class '1'	167	74.17	0.9701	0.9703	0.9801	0.9822
Letter Recognition class 'A'	0	51.8	0.9733	0.9657	0.9645	0.9635
Breast Cancer	16	96.82	0.9693	0.9605	0.9912	0.9561
Banknote Authentification	0	86.01	1.0	1.0	0.9903	0.9963
Page Blocks	6	80.26	0.9409	0.9346	0.9799	0.9698
Monks 2	0	79.88	1.0	0.8786	0.9923	0.9942

shown us that the *DeepRED* implementation where no network or rule pruning is performed extracts more faithful rules when more data is present, yet these rules are much longer and may therefore not be preferable. We consider it therefore a reasonable option to reduce the number of used training instances in order to obtain more comprehensive models.

Additionally, there are two parameters which control the behaviour of the included *C4.5* algorithm by determining when it should stop further growing a decision tree. As an additional general setting we adjust *C4.5* to only perform binary splits and to produce a maximum decision tree depth of ten.

The first parameter – *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.

The network pruning alternatives are also controlled by a parameter, namely the maximum *accuracy decrease*, which is the percentage of the original accuracy to which that of the pruned network may fall. RxREN only prunes inputs as long as the accuracy stays above the accepted value, and stops trying to prune after the first failed attempt. Weight Sparseness Pruning does not prune a connection if the target accuracy is not reached after retraining.

However, as considering different settings of these parameters would have greatly increased the number of experiments, we decided to select only one setting which seemed appropriate for each pruning mechanism based on preliminary experiments. For Weight Sparseness Pruning, we decided to allow a 1% decrease in accuracy, as this already results in pruning a great percentage of connections. For RxREN, we allow a 5% decrease, as this prunes a considerable number of inputs for many datasets, and a higher setting would result in a too great impact on the fidelity of the model. The effect of applying the pruning mechanisms can be observed in Table 4.

As a rule extraction baseline we choose to use the *C4.5* algorithm, that is also used as a part of *DeepRED* itself. For this, we consider two variations regarding the class value used

by the algorithm. One is the regular *C4.5*, which uses the class labels of the original data sets. The other, which we refer to as *FlatRED*, regards the outputs of the network as class values instead of the real classes. It is thus a pedagogical approach for extracting rules from neural networks.

We observe a variant of *FlatRED* for each combination of network and High Fidelity Post-pruning, and two variants of *C4.5*, namely with and without High Fidelity Post-pruning. The introduced *C4.5* parameters apply for all baselines.

5.3 Evaluation Measures

There are two central objectives that we consider for the final rules, which is that they should mimic the behavior of the original neural network, and they should be comprehensible.

We measure the comprehensibility by the number of terms in the extracted rule set to compare *DeepRED* with the baselines. Also, we observe the number of intermediate conditions to gain insight into how this amount varies depending on the network pruning approaches and to whether High Fidelity Post-pruning was applied.

We quantify the second objective with the *fidelity* measure, which is the ratio of unseen instances that are classified by the extracted model the same way as by the original network.

$$fidelity(NN, rules, D_{test}) = 1 - \frac{1}{t}\sum_{i=1}^{t} \Delta(NN(x_i), rules(x_i))$$

where $\Delta(y, y') = 1$ if $y \neq y'$ and 0 otherwise, and $D_{test} = \{x_1, \ldots, x_t\}$ is the example set were the measure is evaluated on.

Additionally, we are interested in observing how the extracted rules fare in terms of predicting the correct class of new instances against models which were trained with the real class labels. For this, we use the accuracy measure.

$$accuracy(rules, D_{test}) = 1 - \frac{1}{t}\sum_{i=1}^{t} \Delta(y_i, rules(x_i))$$

We also want to detect what variant of *DeepRED* is more appropriate for different positions on the trade-off curve between complexity and predictive quality, and how it compares against the baselines at these placements. As these two objectives are not normalized and the relative costs between them may vary with respect to the dataset and to the user's needs, we calculate the *Pareto frontier*, which is a framework for defining a strict ordering of options with regards to conflicting goals.

In this framework, a configuration v strictly dominates w (or $v \succ w$) if $v_i \leq w_i$ for every objective loss i and at least one inequality is strict. That is to say, if v is at least as good as w for all goals, and better for at least one. All options which are not strictly dominated are considered to be *Pareto optimal* and placed on the Pareto frontier. Note that in our case the objectives are to maximize the fidelity and to minimize the number of terms.

5.4 Evaluation Setup

Before we setup the evaluation, we collected our expectations of *DeepRED*'s performance. Our expectations were:

1. *DeepRED* can profit from a NN's structure when the data models a concept that cannot be described or learned well by a simple decision tree. In such cases, using *DeepRED* is a way to obtain a model with a rule representation which maintains the higher predictive accuracy of the neural network.
2. For problems where there is a great number of input attributes and no previous feature selection or preprocessing has been performed, RxREN pruning helps to extract shorter rules than those extracted with no previous RxREN pruning that have a similar fidelity.
3. Weight Sparseness Pruning encourages *DeepRED* to extract more abstract hidden features, which results in final rules with less number of terms that are more faithful to the network with regards to unseen instances.
4. Applying High Fidelity Post-pruning considerably decreases the number of terms of the extracted rules while maintaining a higher accuracy than if the complexity was only controlled via the C4.5 parameters.

As there are many aspects we want to explore, we decided to perform the experiments in three different phases. In Section 6.1, it is explored to what extent *DeepRED* can extract rules from networks that model logical concepts and whether using Weight Sparseness Pruning is profitable in this regard. In Section 6.2, different variants of *DeepRED* are compared using six datasets. Finally, in Section 6.3, we compare how the Pareto optimum configurations of *DeepRED* fair against those of the baselines.

6 Evaluation

In the following, we will start with the discussion of results on the reconstruction of logical function from neural networks, which served us as a motivation for the development of some of the introduced techniques. In the sensitivity analysis in Section 6.2 we will try to better understand the behaviour of *DeepRED* and its parameters. Eventually, we will present the reader the results of the comparison with the baselines with a emphasis on trading off fidelity and comprehensibility (cf. Section 6.2.3).

6.1 Reconstruction of logical functions

Deep neural networks are able to model complex logical patterns in their intermediate layers. Discovering these subconcepts is essential to predicting how the network would respond to new instances, as well as to gain insight into how the classification process works. To successfully find such patterns it is paramount to observe what occurs between the input and output layers. In the following section we explore to what extend *DeepRED* is successful at finding hidden logical concepts in networks trained to emulate boolean functions.

For these experiments, no post-pruning was used, as the goal was to observe what concepts were extracted without this varying factor, and preliminary experiments showed that applying post-pruning when extracting rules from networks which modeled boolean expressions had an inconsistent effect, increasing fidelity in some cases and lowering it in a few, but usually not altering it.

For rule extraction, the data was split into ten, four or two stratified folds and a cross-validation was performed, respectively. In a second run train and test sets were inverted, resulting in experiments with 10%, 25%, 50%, 75% and 90% of the data being available for rule extraction. Regarding the stopping criteria of C4.5, 97% class dominance and 1% dataset size were used. Note that in the case of the boolean functions, *FlatRED* corresponds

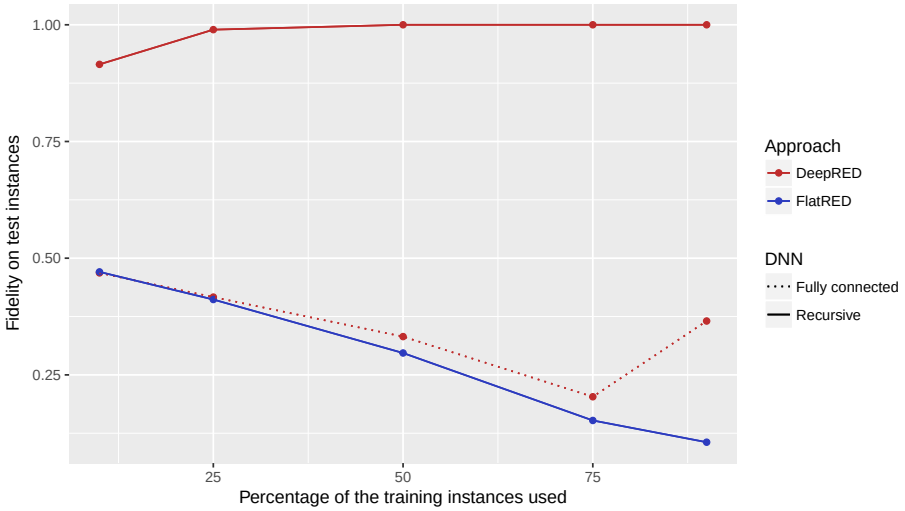

Fig. 9 Fidelity of applying *DeepRED* and *FlatRED* to manually constructed and trained NN representing the parity function.

to C4.5 as the accuracy of the networks is 100%. In the same manner, *FlatRED* is not able to distinguish between manually constructed and trained NN representing the parity function.

We first extracted rules from the artificial and trained networks without any pruning. What we found was that *DeepRED* was very successful at extracting rules from the manually constructed network, needing only a small subset of the possible boolean combinations to reconstruct the original expression. Yet, the same did not hold for the trained network, where *DeepRED* performed similarly to *FlatRED*. As can be seen in Figure 9, when extracting rules from this network, the accuracy on the test set never exceeded fifty percent, and having greater amounts of data at disposal decreased instead rather than improved the fidelity.

This result is quite intriguing since predicting always the any of the two classes, or randomly, achieves an accuracy of 50%. One could argue that even if C4.5 learned a random model and not the default rule, the accuracy should be close to 50%. We will try to explain this phenomenon by means of the following example: suppose that you learn the XOR-3 function, and your train split contains 000, 011 among others, but not 001 and 010. The learner will learn the rule "IF $x_1 = 0$ THEN 1" since it will have enough evidence for it. However, this rule will achieve an accuracy of 0% on the test set containing 001 and 010. For XOR-8, this effect starts to happen as soon as training examples are added and increases the more training instances are available. If we performed leave-one-out cross validation, we would very likely obtain an accuracy of 0%.

The neurons in the first layer of the trained NN apparently model similarly “unstable" functions depending on a high number of inputs, preventing *DeepRED* from generalizing correctly. In the manually crafted NN we also have XOR functions at the first layer. However, DeepRED quickly obtains its evidence since it only has to observe four different combinations of two inputs.

Applying Weight Sparseness Pruning considerably reduces the number of connections, simplifying the internal structure. We wanted to explore if this also resulted in more abstract intermediate concepts being learned, which would facilitate the latter rule extraction when few examples were present.

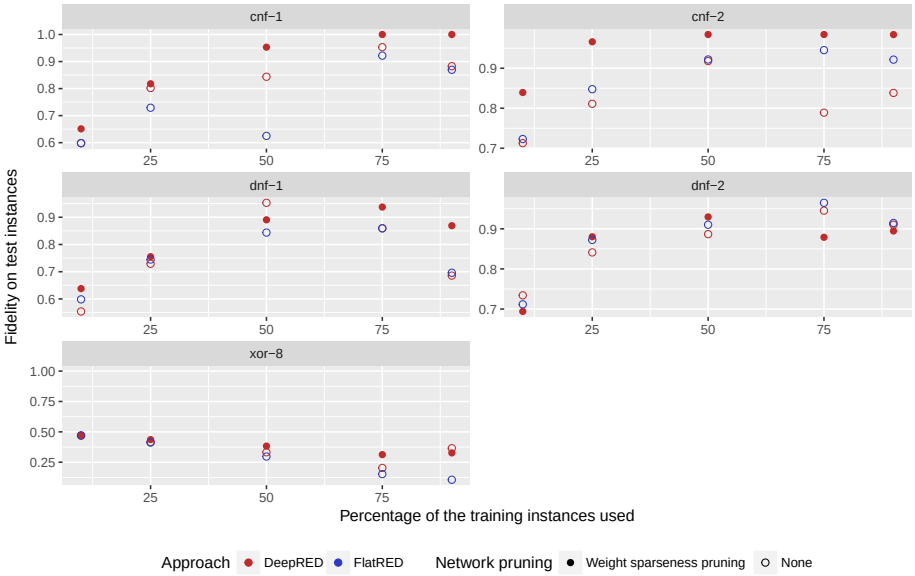

Fig. 10 Fidelity of *FlatRED* and *DeepRED* w.r.t. ratio of training instances used and weight sparseness network pruning (1%). *FlatRED* is applied only on unpruned NN due to the small difference in the training signal (max. 1% different outputs).

As can be seen in Figure 10, this did indeed have in most cases a very positive effect on the fidelity, and allowed *DeepRED* to extract faithful rules when it otherwise would not have been capable to. However, in the case of the XOR function, the performance remained highly unsatisfactory and did not come close to that of the rules extracted from the manually designed network.

When analyzing the expression extracted from the XOR networks we noticed that *DeepRED* starts building intermediate concepts from the manually constructed network from the deeper layers, following the recursive structure. Yet, when presented with the trained network it concentrates the logic on the shallower layers. This happened again, although to a lesser degree, with the network that had been subjected to Weight Sparseness Pruning. Table 5 shows the number of conditions considered for each hidden layer.[6]

6.2 Sensitivity Analysis of *DeepRED* variants

We selected six datasets to perform a more thorough analysis on how the different modifications of *DeepRED* assisted the approach in order extract more faithful and/or comprehensible rules. We were also interested in the effect of the C4.5 parameters and using different amounts of data overall, as well as to how the variants coped with these varying factors. All configurations of *DeepRED* are summarized in Table 6.

We performed experiments with a total of four different C4.5 stopping criteria. However, we did not attempt the configuration of 99% class dominance and no restriction on dataset size for the *artificial 1* and *artificial 2* datasets, as we deemed that by not considering the

[6] Note that decimal values are present as the values are averaged over the corresponding data splits.

Table 5 Number of total and intermediate conditions extracted from each network modeling the XOR function when presented with 90%, 50% and 10% of all possible combinations.

DNN's construction method	Percentage of examples used	# terms in final expression	# intermediate terms extracted per hidden layer
Manual	90%	1024	16.0 - 8.0 - 8.0 - 4.0 - 2.0
Trained	90%	249	8.0 - 6.0 - 2.0 - 1.0 - 1.0
Trained, pruned	90%	383.25	22.25 - 7.0 - 2.0 - 2.0 - 2.0
Manual	50%	1024	16.0 - 8.0 - 8.0 - 4.0 - 2.0
Trained	50%	154	8.5 - 6.5 - 2.0 - 1.0 - 1.0
Trained, pruned	50%	312.5	31.5 - 11.0 - 3.5 - 3.0 - 3.0
Manual	10%	822	16.25 - 9.0 - 8.0 - 4.0 - 2.0
Trained	10%	34.5	4.5 - 4.0 - 1.25 - 1.0 - 1.0
Trained, pruned	10%	38.75	7.0 - 4.25 - 1.25 - 1.25 - 1.25

Table 6 The parameter settings used for the evaluation.

C4.5 parameters
92% / $\leq$ 3%
95% / $\leq$ 2%
97% / $\leq$ 1%
99% / $\leq$ 0%

Network pruning
No pruning
RxREN pruning 5%
Weight Sparseness Pruning 1%

Pruning of rules
No pruning
High Fidelity Post-pruning

Training set
10%
25%
50%
100%

size of the original dataset when choosing to stop branching this would be problematic for this larger data sets.

As we wanted to observe how each combination performed when using different amounts of data, rules were extracted with 10%, 25%, 50%, 75% and 100% of the train instances. To determine what data to use, the train set was split into ten, four or two stratified folds and, as done in Section 6.1. However, only four different combinations of train and test folds of the 10-fold cross-validation were randomly chosen for averaging.

Additionally, each test set was split into two stratified, equally large subsets. One was regarded as the validation set upon which the performance of the different approaches were examined and to calculate which configurations would fall in the Pareto front, and the other half was used to compare the best variants against the best variants for each baseline, chosen as well with the validation set.

6.2.1 Effect of network and High Fidelity Post-Pruning on the comprehensibility

We first examine the impact the different alterations to the base *DeepRED* implementation have on the comprehensibility of the final expressions. Figure 11 illustrates how the total number of conditions in the intermediate expressions, that is to say, the total number of conditions in all rules between any two layers, correlates with the number of conditions in the final rule set. It is clearly shown that applying High Fidelity Post-pruning (filled symbols) keeps the number of final terms from exceeding the first hundred, even when there are more than thousand intermediate terms.

Another interesting observation is that the number of final terms almost always is lower than the number of intermediate rules before resolving although the naive expectation would

be that it should be higher or even explode.[7] These results seem to confirm the general assumption that trained NN are highly redundant. Moreover, it demonstrates the effectiveness of the proposed techniques in order to remove unsatisfiable and redundant rules.

In Figure 13, the number of final conditions when applying the two types of network pruning is compared to the corresponding configurations when no pruning was applied. Only configurations were considered where High Fidelity Post-pruning was performed. The plot shows that post-pruning usually decreases the number of final terms obtained. However, the effect is less pronounced for weight spareness pruning, for which a certain number of cases exists, where the complexity slightly increases.

6.2.2 Finding the optimal DeepRED configuration for a specific trade-off

First, we take a look at the fidelity of the extracted rules in the train and validation sets, visualized in Figure 14. For the *Artificial* datasets, *Monks 3* and *Wilt*, there seems to be a linear correlation with the fidelity on the validation set laying slightly under the fidelity on the train set. For *Monks 1* and *Balance Scale*, there is a much stronger variance regarding the generalization abilities of the different configurations, and also of the same configurations when using different amounts of training data, which is the factor that is not differentiated in the plot.

Figure 12 explores the changes caused by applying High Fidelity Post-pruning, Weight Sparseness Pruning and RxREN and altering the C4.5 stopping criteria irrespective of the amount of data available for rule extraction. For each data availability value, the base configuration with the laxest stopping criteria (92% class dominance and 3% database size) was placed at the origin, and all variations from this configuration were placed on the position determined by their difference in terms of number of final terms and fidelity.

We notice that applying High Fidelity Post-pruning not only significantly reduces the number of final terms, but that it does this while having no or a slight positive effect on the fidelity. We also see that rules extracted from networks subjected to Weight Sparseness Pruning are generally the most faithful to the original network, despite the 1% decrease in accuracy of the networks. Laxer C4.5 stopping criteria positively correlate with the fidelity, but also with the complexity of the model. However, the effect on the number of final terms occurs to a much lesser degree than when High Fidelity Post-pruning is applied.

In Figure 15 we observe the position with regards to the mentioned trade-off only for those configurations where High Fidelity Post-pruning was applied. We see again that for most datasets, applying Weight Sparseness Pruning had a very positive effect on the fidelity. We also notice that *Balance Scale* is the only dataset where laxer C4.5 stopping criteria causes overfitting and thus a drop in fidelity. It seems that when mimicking the original network is the goal to optimize it is preferable to allow C4.5 to keep branching until this causes no improvement.

Finally, we take a look at the configurations which are Pareto optimal for the trade-off of fidelity to comprehensibility, shown in Figure 16. We see that in the end of the spectrum that optimizes fidelity there are mostly configurations that use Weight Sparseness Pruning, whereas when comprehensibility is preferred configurations using RxREN or no network pruning performed better. For *Monks 1*, using RxREN pruning led to the best rules for all preferences. We also see that except for the *Balance Scale* dataset, for which we observed

[7] For instance, representing XOR with 8 inputs would at least require 14 intermediate rules with 2 terms each, but 128 conjunctions with 8 terms each if we wanted to represent it as flat set of rules.

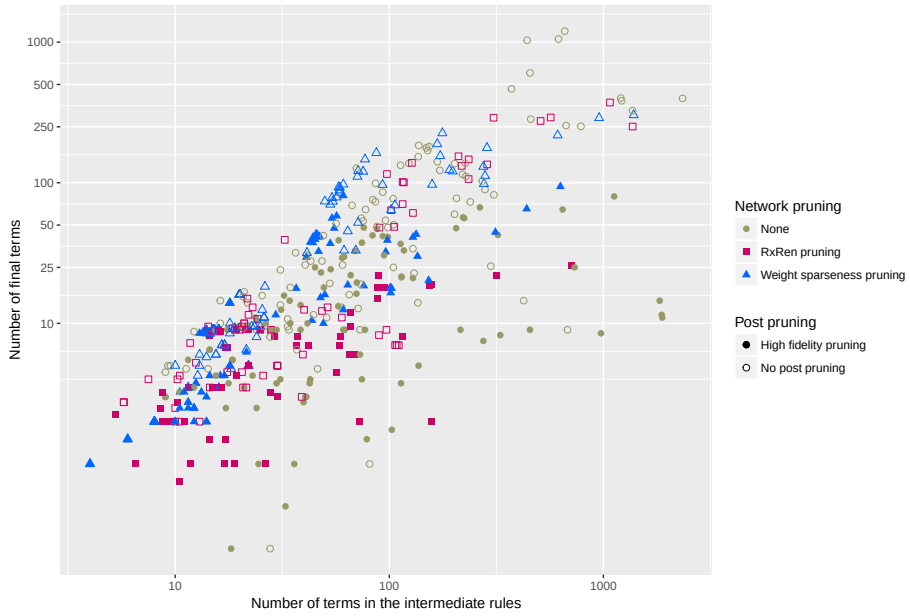

Fig. 11 Relation between number of terms in the intermediate rules and number of terms in the final rules, using High Fidelity Post-pruning and without. Each point corresponds to one parameter combination (pre-pruning, C4.5 control parameters, size of training set) and one dataset. Points were excluded which consist of a default rule.

that High Fidelity Post-pruning considerable decreased accuracy, the majority of the optimal configurations apply this post-pruning method.

6.2.3 Comparison with the baselines

We now observe how the configurations that fall in the Pareto frontier for the validation set fare against those that are Pareto optima for the baselines. We first compare *DeepRED* with the variants of the pedagogical *FlatRED* approach, which considers the classifications of the neural network as class data. For both approaches, the configurations were taken that are Pareto optimum for the validation set in having higher fidelity and less terms. The performance of these on the test set is illustrated in Figure 17. Although *DeepRED* is successful at finding more faithful rules than the pedagogical approach, the improvement margin is not great for most datasets, and *FlatRED* tends to find smaller models.

We also explored whether *DeepRED* could act as an alternative way of finding accurate rule models. For this, we calculated the Pareto optima configurations for *DeepRED*, *FlatRED* and C4.5 on the validation set when optimizing accuracy instead of fidelity, as well as a small number of terms. This comparison can be observed in Figure 18. The results are not very different, as *FlatRED* performed very similarly to C4.5 and there was not much change between optimizing fidelity and accuracy.

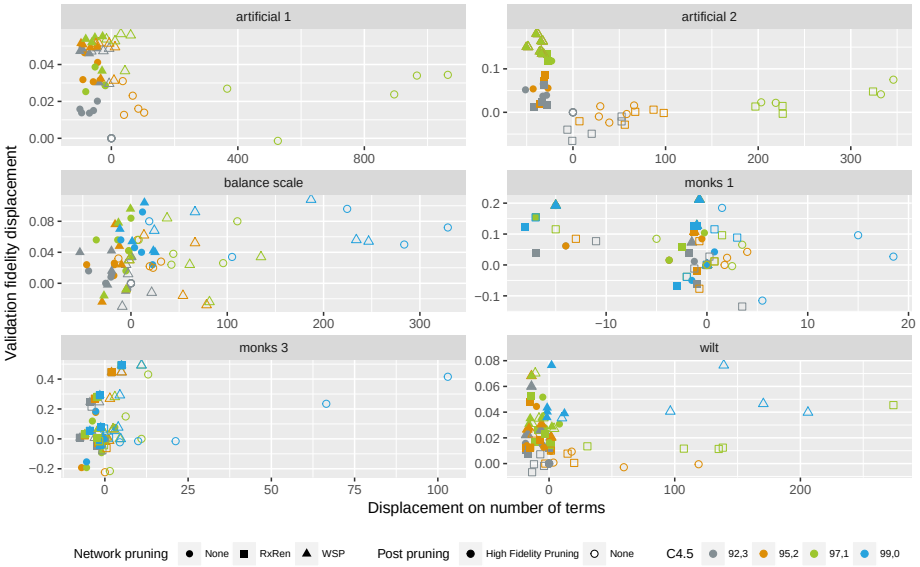

Fig. 12 Effect of different parameters and pruning mechanisms when varying amounts of data are present. For each group of experiments that used the same percentage of training data, the configuration for which the C4.5 parameters 92% and 3% were used and no network pruning or High Fidelity Post-pruning took place was placed at the origin. Each member of the group was then positioned according to its displacement from the it.

6.3 Comparison of the remaining datasets with the baselines

In the last set of experiments, we extracted rules for our last six datasets. We also only used the 99% class dominance with no restriction on minimum dataset size configuration of the C4.5 stopping criteria for the datasets *Breast Cancer*, *Banknote Authentification* and *Monks 2*. The data was split as in the experiments described in Section 6.2, and again a maximum of four repetitions were performed for the experiments using 10% of the train set.

In Figure 19 we observe the configurations which fall in the Pareto frontier on the validation set for optimizing the trade-off between fidelity and number of terms. We again see a predominance for configurations which used a greater percentage of the train data, as well as increasing preference for the laxer C4.5 stopping criteria as the preference for fidelity increases. For the *Letter Recognition A*, *MNIST 1*, *Monks 2* and *Breast Cancer* datasets, the configurations with Weight Sparseness Pruning performed best, whereas this does not hold for the remaining two datasets.

RxREN pruning was only able to reduce the inputs on the datasets *MNIST 1*, *Breast Cancer* and *Page Blocks*, so its effect should only be considered on these plots. For the first two, we see that these configurations made an appearance on the lower end of the spectrum. We had expected particularly the *MNIST 1* dataset to take advantage of this pruning given the high percentage that could be pruned with a small decrease on accuracy. However, it is worth noting that the Weight Sparseness Pruning also indirectly prunes inputs.

Figure 20 compares the configurations of *DeepRED* and *FlatRED* that dominate, respectively for each approach, the fidelity to comprehensibility trade-off on the validation set. However, the performances of the configurations are shown on the test set, reproducing

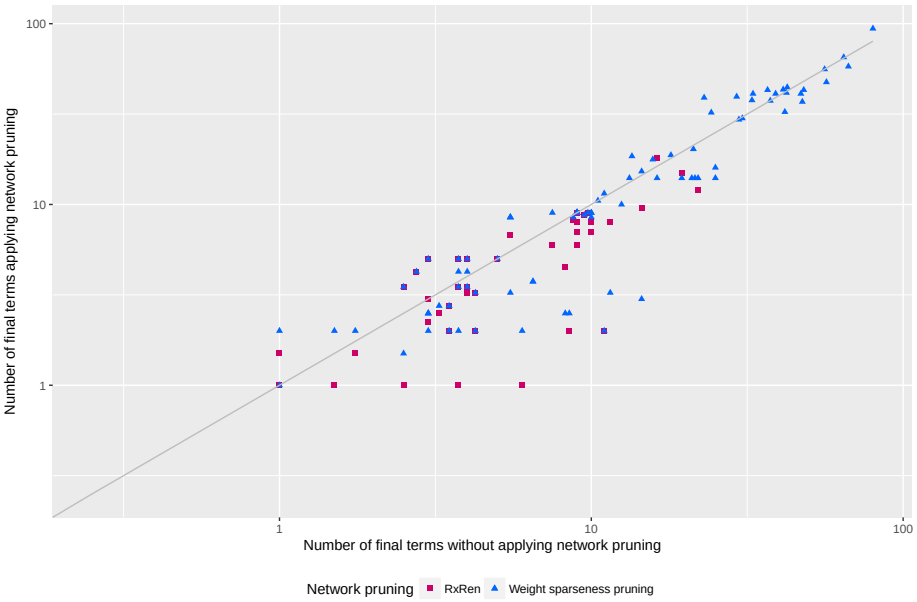

Fig. 13 Relation between number of terms in the final rules if no pre-pruning and if either RxREN or Weight Sparseness Pruning was performed. Each point corresponds to one parameter combination (C4.5 control parameters, size of training set) and one dataset. High Fidelity Post-pruning was always turned on. Points were excluded for which RxREN pruning did not prune any input, or for which the model of at least one of the corresponding folds is a default rule.

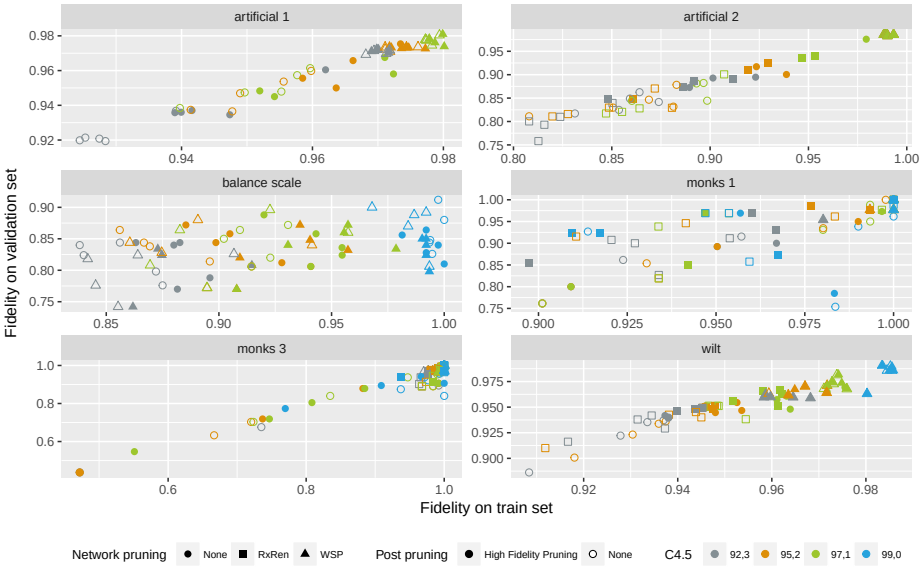

Fig. 14 Relation between the fidelity towards the train and validation sets, illustrating the generalization performance of the extracted models. Each point corresponds to one parameter combination (C4.5 control parameters, size of training set) and one dataset.

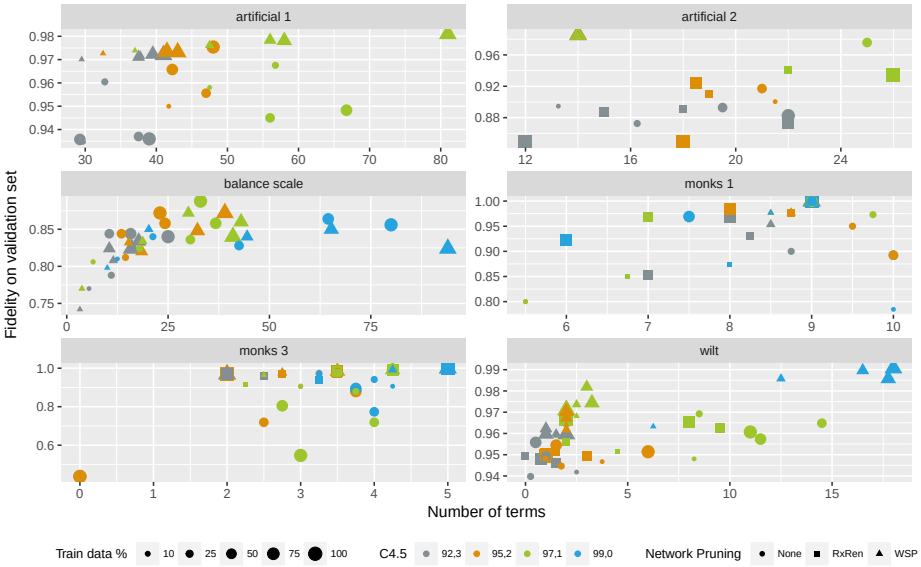

Fig. 15 Relation between number of terms and fidelity on the validation set for *DeepRED* variants using High Fidelity Post-pruning. Each point corresponds to one parameter combination (pre-pruning, C4.5 control parameters, size of training set) and one dataset.

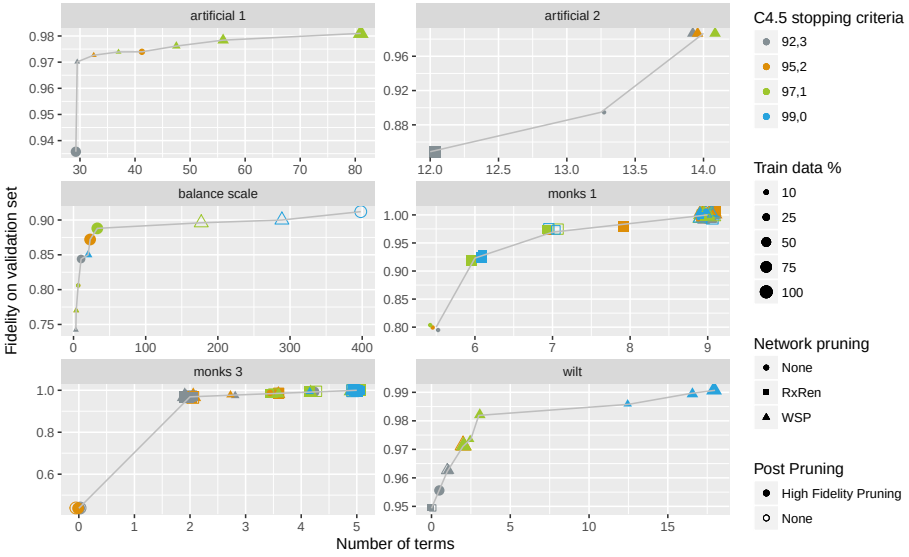

Fig. 16 Configurations of *DeepRED* that fall in the Pareto frontier maximizing fidelity and minimizing number of conditions in the validation set for first six datasets in Table 1. A 0.01 jitter was applied on the datasets *artificial 1*, *monks 1* and *monks 2*. The gray line joining the points shows the true position.

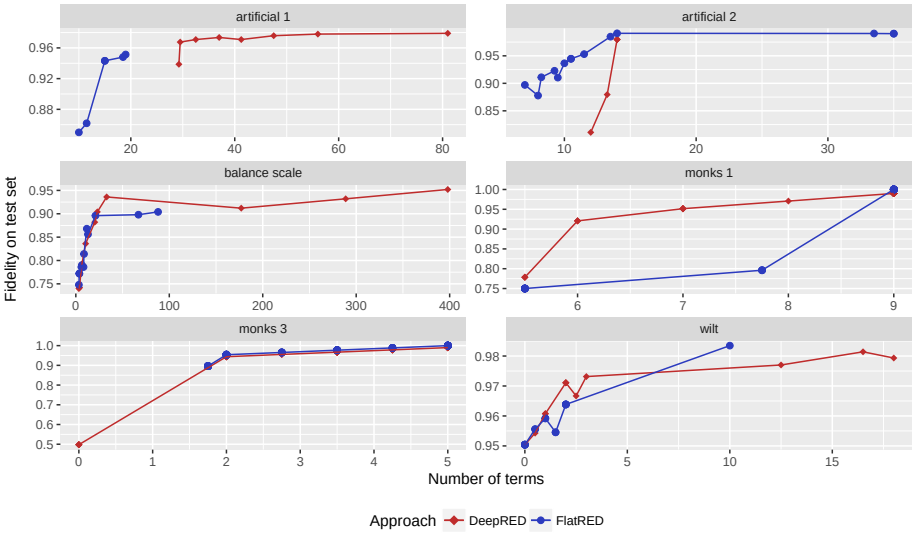

Fig. 17 Comparison between *DeepRED* and *FlatRED* regarding the trade-off between model complexity and fidelity on six datasets. The graphs show the results on the test set for the parameter combinations which were on the Pareto front on the validation set. For the datasets *artificial 2*, *monks 1* and *monks 3*, a jitter of -0.01 on the *y* axis was added for the *DeepRED* points.

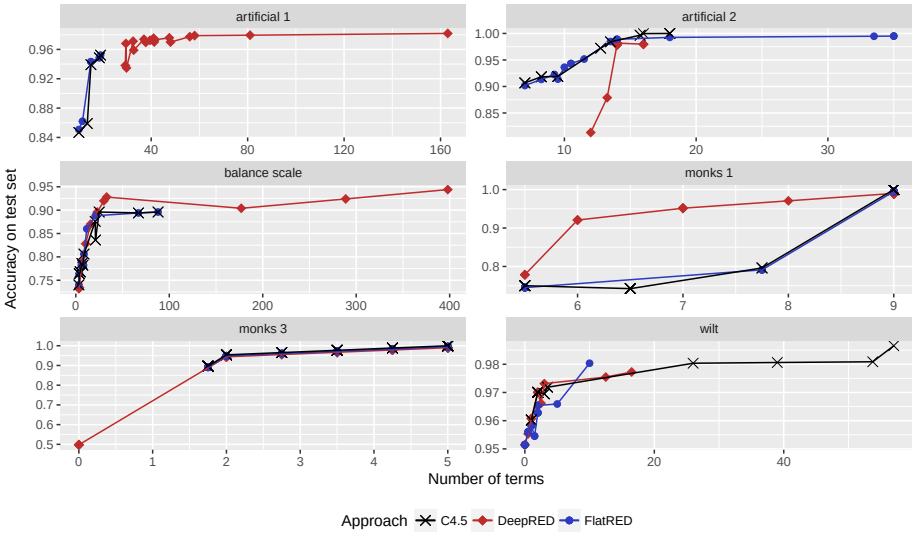

Fig. 18 Comparison between *DeepRED*, *FlatRED* and *C4.5* regarding the trade-off between model complexity and accuracy on six datasets. The graphs show the results on the test set for the parameter combinations which were on the Pareto front on the validation set. For the datasets *artificial 2*, *monks 1* and *monks 3*, jitters of -0.01 and -0.005 on the *y* axis were added for the *DeepRED* and *FlatRED* points, respectively.

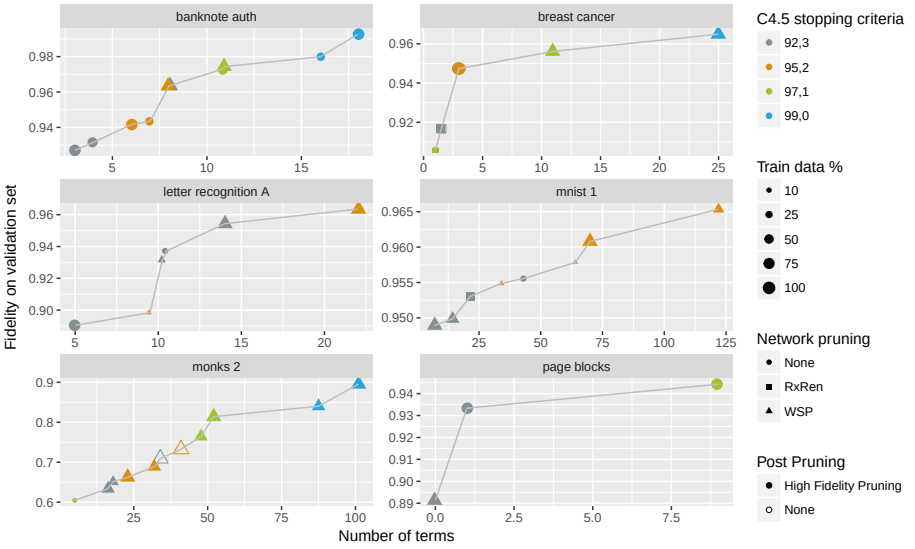

Fig. 19 Configurations of *DeepRED* that fall in the Pareto frontier maximizing fidelity and minimizing number of conditions in the validation set for the six last datasets in Table 1.

a real situation where the test set is unknown and the best configuration according to the user's needs has to be chosen on a hold-out set. For most datasets, *DeepRED* was successful at extracting more faithful rules, however for the positions in the trade-off curve that favor comprehensibility, *FlatRED* seemed to outperform *DeepRED*, especially for *Letter recognition* and *MNIST*. Here, also, a considerable variance can be noticed between the performance of some configurations in the validation set and in the test set.

Figure 21 compares the accuracy of *DeepRED* to both *FlatRED* and *C4.5*, constructed similarly as for the illustration of the fidelity – comprehensibility trade-off. Both *FlatRED* and *C4.5* seemed to outperform *DeepRED* in terms of comprehensibility, and *DeepRED* only achieved a considerably higher accuracy for the datasets *Monks 2* and *MNIST 1*. Particularly for *MNIST 1*, the pedagogical approaches seemed unable to achieve a high fidelity. This is only the only dataset considered with large amount of unprocessed input features, which DNNs are well known to handle particularly well.

It is worth noting than on the *Monks 2* dataset, applying Weight Sparseness Pruning had the unexpected effect of considerably increasing the accuracy of the network from 0.8786 to 0.9942 for the combined validation and test sets, whereas there was not much change on the train set. This is why the highest achieved accuracy is much higher than the highest achieved fidelity to the original network. It appears that *DeepRED* was able to capture the modifications in the network that caused the accuracy improvement, whereas this did not occur for the *FlatRED* variant which extracted rules from the pruned network.

On the other hand, in *Page Blocks* the accuracy was reduced from 0.9896 to 0.9698, likely also reducing the fidelity of the pruned network to the original one. This could explain why no variant using Weight Sparseness Pruning fell in the Pareto front. These effects can be observed in Table 4.

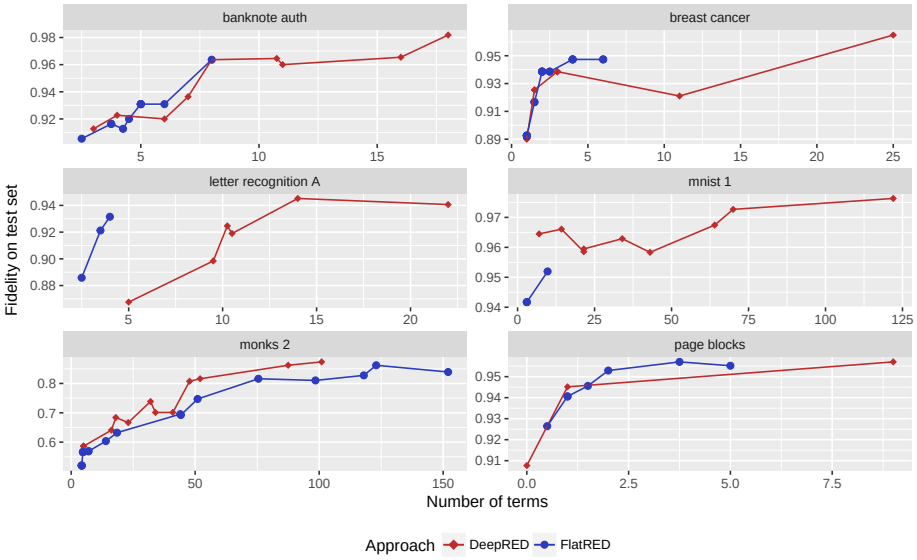

Fig. 20 Comparison between *DeepRED* and *FlatRED* regarding the trade-off between model complexity and fidelity on six datasets. The graphs show the results on the test set for the parameter combinations which were on the Pareto front on the validation set.

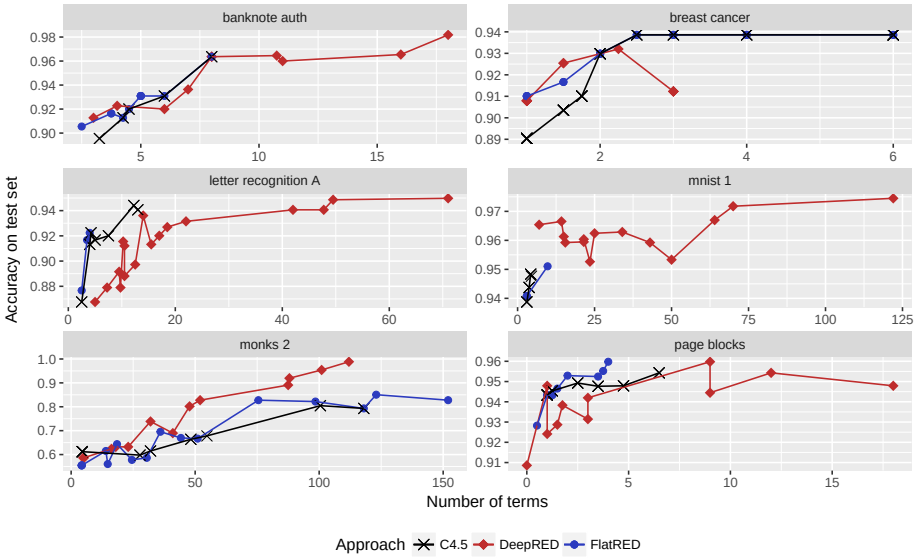

Fig. 21 Comparison between *DeepRED*, *FlatRED* and *C4.5* regarding the trade-off between model complexity and accuracy on six datasets. The graphs show the results on the test set for the parameter combinations which were on the Pareto front on the validation set.

7 Discussion and Conclusions

In this paper, we presented and evaluated *DeepRED*, a novel decompositional algorithm that solves the problem of extracting rules from *deep* neural networks. Our motivation was two-fold: on the one hand, we intended to make the internal decision processes of a deep neuronal network more comprehensible and transparent to humans. This involves understanding the logic which the NN use internally to make its predictions, represented as a set of comprehensible rules, as well as gathering insights into the internal distribution of the decision process among layers and neurons in order to, for instance, explain and identify the source of a particular prediction. On the other hand, we expected that observing the internal structure of the DNN may uncover hidden features, thus allowing the extracted rules to better mimic the original network as well as to find better and shorter representations. This would hopefully also allow to find accurate symbolical classification models for tasks for which state-of-the-art rule or decision tree learners underperform. A prototypical, although extreme, example used throughout this work was the parity function, for which common approaches hardly can find a solution.

DeepRED extends the *CRED* algorithm, defined for neural networks with one hidden layer. Our approach uses *C4.5* to create rules that describe neurons on the basis of neurons in the preceding layer. The rules can then be merged to produce a rule set that mimics the overall behavior of the given DNN.

Several modifications were made to the way the intermediate and final rules are simplified and pruned. Extending the *CRED* approach to a higher number of layers has on the one hand the effect that the final number of terms may become much higher, yet it also means that the error propagated from the proceeding layer should be contained.

We thus extend the logical simplification to target a pitfall typical of extracting rule sets from decision trees. The proposed High Fidelity Post-Pruning approach for rules was able to effectively reduces the size of the resulting rule set without compromising the fidelity (Section 6.2).

Our expectations for *DeepRED* to find more accurate and shorter representation by distilling the knowledge of the NN was proven correct to a certain extent, yet it was shown that the ability of *DeepRED* to extract such concepts depends highly on the internal structure of the network. For instance, *DeepRED* showed to be very effective in mimicking a manually crafted deep network representing the XOR-function, but failed similarly to the baselines when learning the structure of a trained network. The inspection of the extracted intermediate rules suggested the cause in the concentration of the NN's logic in the first hidden (just as it would be for a shallow network), making it impossible for DeepRED to benefit from deeper structures.

To encourage such structures we introduce the Weight Sparseness Pruning algorithm, which prunes network connections so that each neuron is only connected to a limited number of neurons from the surrounding layers. This step considerably improved the fidelity of the rules extracted from networks modeling boolean functions, as well as better mimicking the behavior of several networks trained with artificial and real datasets, often outperforming the baselines and the *DeepRED* variants that made no use of this modification.

Rules extracted with *DeepRED* were not consistently more accurate than those built directly from the train data. However, this did prove to be the case in some cases, and it is worth noting that we did not pay special attention into the training of the DNNs to maximize their accuracy. Particularly in the case of the *Monks 1* dataset, where applying Weight Sparseness Pruning had the unexpected effect of considerably increasing the accuracy of the

network on the test data, *DeepRED* was able to leverage these hidden concepts to outperform the baselines.

As far as the comprehensibility of the final expressions, *DeepRED* learning intermediate features in most cases did not result in more compact final expressions. However, the size of the concepts learned by *DeepRED* with turned on High Fidelity Post-Pruning was usually on par with the baseline. It is also shown that the fidelity to comprehensibility trade-off can be effectively controlled through the C4.5 stopping criteria.

It is also worth noting that *DeepRED* can be effectively be used to provide meaningful insights into the internal structure of the neural network, for instance through the number of terms in the intermediate expressions.

Despite several effective counter-measures for reducing the size of the resulting rules, especially during the delicate merging of the rules for two subsequent layers, the high computational costs in some scenarios are still a limitation for the proposed methodology. The main bottlenecks are the quadratic number of weights between layers with a high number of nodes, and in some cases the number of training examples due to a strong increase in the number of split candidates which the decision tree learner has to consider. The usage of more efficient discretization methods already showed to solve the latter problem, but further improvements could be achieved by more intelligent splitting techniques taking into consideration the neuron at hand. We leave these directions for further research.

The problem of the quadratic number of weights could already be alleviated by the introduced Weight Sparseness Pruning. However, as our inspection of the intermediate rules have shown, further research on training sparse and better balanced networks (throughout the depth) seems worthwhile.

References

Proceedings of NIPS 2016 Workshop on Interpretable Machine Learning for Complex Systems (Nov 2016)

Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016) (Jul 2016)

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

Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. arXiv preprint arXiv:1512.08571 (2015)

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

Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 2654–2662. Curran Associates, Inc. (2014), http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one 10(7), e0130140 (2015)

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

Bucila, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006. pp. 535–541 (2006)

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., Elhadad, N.: Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015. pp. 1721–1730 (2015)

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

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

Craven, M.W., Shavlik, J.W.: Extracting tree-structured representations of trained networks. Advances in neural information processing systems pp. 24–30 (1996)

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

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

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

Gallego-Ortiz, C., Martel, A.L.: Interpreting extracted rules from ensemble of trees: Application to computer-aided diagnosis of breast mri. arXiv preprint arXiv:1606.08288 (2016)

Goodman, B., Flaxman, S.: European union regulations on algorithmic decision-making and a" right to explanation". arXiv preprint arXiv:1606.08813 (2016)

Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. pp. 1135–1143 (2015)

Hassibi, B., Stork, D.G., et al.: Second order derivatives for network pruning: Optimal brain surgeon. Advances in neural information processing systems pp. 164–164 (1993)

Hinton, G.E., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. CoRR abs/1503.02531 (2015), http://arxiv.org/abs/1503.02531

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

Johnson, B.A., Tateishi, R., Hoan, N.T.: A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. International journal of remote sensing 34(20), 6969–6982 (2013)

Kamruzzaman, S., Hasan, A.R.: Rule extraction using artificial neural networks. arXiv preprint arXiv:1009.4984 (2010)

Kasneci, G., Gottron, T.: Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. pp. 45–54. ACM (2016)

Kayande, U., De Bruyn, A., Lilien, G.L., Rangaswamy, A., Van Bruggen, G.H.: How incorporating feedback mechanisms in a dss affects dss evaluations. Information Systems Research 20(4), 527–546 (2009)

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

LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November

27-30, 1989]. pp. 598–605 (1989), http://papers.nips.cc/paper/250-optimal-brain-damage
Lichman, M.: Uci machine learning repository, 2013. URL http://archive. ics. uci. edu/ml 8 (2015)
Lipton, Z.C.: The mythos of model interpretability. CoRR abs/1606.03490 (2016), http://arxiv.org/abs/1606.03490, presetnted at 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)
Quinlan, J.R.: Generating production rules from decision trees. In: IJCAI. vol. 87, pp. 304–307. Citeseer (1987a)
Quinlan, J.R.: Simplifying decision trees. International journal of man-machine studies 27(3), 221–234 (1987b)
Quinlan, J.R.: C4.5: Programs for Machine Learning, vol. 1. Morgan Kaufmann (1993)
Ribeiro, M.T., Singh, S., Guestrin, C.: Why should i trust you?: Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1135–1144. ACM (2016)
Russell, S., Norvig, P.: Artificial intelligence: a modern approach. Pearson Education (1995)
Saito, K., Nakano, R.: Extracting regression rules from neural networks. Neural networks 15(10), 1279–1288 (2002)
Sato, M., Tsukimoto, H.: Rule extraction from neural networks via decision tree induction. In: Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on. vol. 3, pp. 1870–1875. IEEE (2001)
Schmitz, G.P., Aldrich, C., Gouws, F.S.: ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. Neural Networks, IEEE Transactions on 10(6), 1392–1401 (1999)
Sethi, K.K., Mishra, D.K., Mishra, B.: KDRuleEx: A novel approach for enhancing user comprehensibility using rule extraction. In: Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on. pp. 55–60. IEEE (2012)
Setiono, R.: Extracting rules from pruned neural networks for breast cancer diagnosis. Artificial Intelligence in Medicine 8(1), 37–51 (1996)
Setiono, R.: Extracting rules from neural networks by pruning and hidden-unit splitting. Neural Computation 9(1), 205–225 (1997a)
Setiono, R.: A penalty-function approach for pruning feedforward neural networks. Neural computation 9(1), 185–204 (1997b)
Setiono, R., Leow, W.K.: FERNN: An algorithm for fast extraction of rules from neural networks. Applied Intelligence 12(1-2), 15–25 (2000)
Setiono, R., Leow, W.K., Zurada, J.M.: Extraction of rules from artificial neural networks for nonlinear regression. Neural Networks, IEEE Transactions on 13(3), 564–577 (2002)
Setiono, R., Liu, H.: Symbolic representation of neural networks. Computer 29(3), 71–77 (1996)
Setiono, R., Thong, J.Y.: An approach to generate rules from neural networks for regression problems. European Journal of Operational Research 155(1), 239–250 (2004)
Sharif, M., Bhagavatula, S., Bauer, L., Reiter, M.K.: Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (Oct 2016)
Šíma, J.: Neural expert systems. Neural networks 8(2), 261–271 (1995)
Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
Taha, I.A., Ghosh, J.: Symbolic interpretation of artificial neural networks. Knowledge and Data Engineering, IEEE Transactions on 11(3), 448–463 (1999)

Tang, S., Han, J.: A pruning based method to learn both weights and connections for LSTM. Tech. rep., Stanford University (2015), https://nlp.stanford.edu/courses/cs224n/2015/

Thimm, G., Fiesler, E.: Pruning of neural networks. Tech. rep., IDIAP (1997)

Thodberg, H.H.: Improving generalization of neural networks through pruning. International Journal of Neural Systems 1(04), 317–326 (1991)

Thrun, S.: Extracting provably correct rules from artificial neural networks. Tech. rep., University of Bonn, Institut für Informatik III (1993)

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

Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S.E., Fisher, D., et al.: The monk's problems a performance comparison of different learning algorithms (1991)

Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) pp. 267–288 (1996)

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

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

Turner, R.: A model explanation system. In: Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on. pp. 1–6. IEEE (2016)

Urban, G., Geras, K.J., Kahou, S.E., Aslan, Ö., Wang, S., Caruana, R., Mohamed, A., Philipose, M., Richardson, M.: Do deep convolutional nets really need to be deep and convolutional? In: Proceedings of the International Conference on Learning Representations 2017 (2017), http://arxiv.org/abs/1603.05691

Vapnik, V., Vashist, A.: A new learning paradigm: Learning using privileged information. Neural networks 22(5), 544–557 (2009)

Varshney, K.R., Alemzadeh, H.: On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. arXiv preprint arXiv:1610.01256 (2016)

Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision. pp. 818–833. Springer (2014)

Zeng, J., Ustun, B., Rudin, C.: Interpretable classification models for recidivism prediction. Journal of the Royal Statistical Society: Series A (Statistics in Society) (2016)

Zhou, Z.H., Chen, S.F., Chen, Z.Q.: 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. vol. 3, pp. 401–406. IEEE (2000)