Rules Extraction from Neural Networks Trained on Multimedia Data

Matěj Fanta[1], Petr Pulc[2], Martin Holeňa[3]

[1] Faculty of Nuclear Sciences and Physial Engineering, Czech Technical University, Břehová 7, Prague, Czech Republic
[2] Faculty of Information Technology, Czech Technical University, Thákurova 9, Prague, Czech Republic
[3] Institute of Computer Science, Czech Academy of Sciences, Pod vodárenskou věží 2, Prague, Czech Republic

Abstract: Since the universal approximation property of artificial neural networks was discovered in the late 1980s, i.e., their capability to arbitrarily well approximate nearly arbitrary relationships and dependences, a full exploitation of this property has been always hindered by the very low human-comprehensibility of the purely numerical representation that neural networks use for such relationships and dependences. The mainstream of attempts to mitigate that incomprehensibility are methods extracting, from the numerical representation, rules of some formal logic, which are in general viewed as human-comprehensible. Many dozens of such methods have already been proposed since the 1980s, differing in a number of diverse aspects. Due to that diversity, and also due to a close connection of the semantics of extracted rules to the repsective application domain, no rules extraction methods have ever become a standard, and it is always necessary to select a suitable method for the considered domain. Here, rules extraction from trained neural networks is employed for multimedia data, which is an increasingly important but also increasingly complex kind of data. Three particular rules extraction methods are considered and applied to the modalities recognized text data and the speech acoustic data, both of them with different subsets of features. A detailed comparison of the performance of the considered methods on those datasets is presented, and a statistical analysis of the obtained results is performed.

1 Introduction

Despite the usefulness of artificial neural networks (ANNs), however, full exploitation of their universal approximation property [12, 13, 14] has always been hindered by the very low human-comprehensibility of the purely numerical representation that neural networks use to represent relationships and dependencies (in terms of [11], that representation provides a high data fit, but a low mental fit).

The mainstream of attempts to mitigate that incomprehensibility are methods extracting, from such a representation, rules of some formal logic. This paper is concerned about methods that extract rules in the attributive logic using only typical relation symbols. On the other hand, many onther methods extract fuzzy rules. Such methods can be found in the survey articles like [15, 8].

Logical rules are a frequent way of communicating the knowledge between humans, and they are in general viewed as human-comprehensible. From the point of view of knowledge discovery in data, rules extraction from neural networks has to compete with methods for the extraction of logical rules directly from data, most notably with methods relying on various kinds of decision trees [3, 18, 22]. The main reasons why rule extraction from data via the intermediate step of a trained neural network is attractive even in the competition of direct methods are:

- neural networks take into consideration all input variables at the same time, i.e., they perform a multivariate search, not a search in a variable-by-variable manner;
- in the case of data resulting from continuous random variables, evaluation of the neural network deals with data as continuous, however, rules are using conditions which work as a discretization of input values (for comparison, decision trees include discretization of each variable from the very beginning).

Since the 1980s, many dozens of rules extraction methods have been developed for trained neural networks; a good overview can be obtained from the survey papers [1, 7, 16, 24] and the monograph [9]. In [1], it has been proposed to characterize and categorize them according to the following properties:

(i) *expressive power:* in the language of which logic (Boolean / fuzzy, propositional / 1st-order) are the rules expressed;
(ii) *translucency:* whether the rules extraction method takes into account only the input-output mapping learned by the network, or also the activities of hidden neurons;
(iii) *portability:* whether and in which way the rules extraction method requires specific training of the neural network;
(iv) *quality* of the extracted rules;
(v) *computational complexity* of the employed rules extraction algorithm.

Due to that diversity, and also due to a close connection of the semantics of extracted rules to the respective application domain, no rules extraction methods have ever become a standard, and it is always necessary to select a suitable method for the considered domain.

The main factor in the comparison of the rule extraction methods is the quality of the extracted rules. Let us denote $C(D)$ as ground truth classification of the data D, $NN(D)$ as a classification by NN of the data D, $R(D)$ as a classification by extracted rules of the data D, and the size of the data $m = \#D$. The survey paper [7] introduced four quality measures from which we will use three: Accuracy $\frac{\#\{C(D) \cap R(D)\}}{m}$; fidelity $\frac{\#\{NN(D) \cap R(D)\}}{m}$; and comprehensibility which is task-specific therefore more measures are possible, e.g., the number of the extracted rules, the number of antecedents per rule.

To this end, three rules extraction methods have been selected: ANN-DT [21], DeepRED [26], and HypInv [19]. They are applied to two modalities of a multimedia data collection: recognized text data and speech acoustic data, both of them with different subsets of features. A comparison of the performance of the considered rules extraction methods on those four datasets is presented, and statistical analysis of the obtained results is performed. The selected rules extraction methods are briefly reviewed in the next section, and their performance on the considered multimedia data is presented in Section 3.

Experiments on multimedia data provide a proof of concept of applicability of these methods to multi-class data with very high input dimension. Our orientation towards multimedia data is convenient because authors of rule extraction methods do not report experiments with this kind of data, and some modification of their methods are necessary.

2 Selected Methods for Rules Extraction from Neural Networks

2.1 DeepRED

Nowadays, the most promising decompositional method seems to be DeepRED that can be applied to deep neural networks (DNNs), more precisely to multilayer perceptrons (MLPs) of any depth. It was first published in [26], but the author's master thesis [25] describes it in more detail. The core idea of the algorithm is to build decision trees (DTs) on the activations of the considered NN as the input and compose these trees, or rules derived from them, into more complex ones.

We have performed modifications with the algorithm. It originally used rules obtained from a DT. However, the process of merging rules derived from DTs can be better represented using a decision directed acyclic graph (DDAG). A DDAG is adding properties within its structure, which will be advantageous. Not to mention higher memory usage and more difficult evaluation of the rule representation compared with the graph. If we consider the largest DDAG structure with the depth l, i.e., a full-grown tree with $\left(2^l - 1\right)$ nodes, then the same rule set has the size 2^l with l terms each. Besides, evaluation of the tree is done in at most l decisions, but rule evaluation may need to evaluate all 2^l rules in the worst-case scenario. We have on achieved on the benchmark dataset MNIST results similar to those of the author of the original version of DeepRED.

Let us introduce the following notation: L is the number of layers in the MLP; H_l equals to the number of neurons in the l-th layer; $h_l(X)$ denotes a list of activations of the l-th layer for samples in data X; $t(Y)$ denotes the division of the data Y by the split t, i.e., the class $True$ for samples which fulfilled the t and the class $False$ for others. A pseudocode of the algorithm is shown in Algorithm 1. In the first step, the initialization of the DDAG that provides a mapping from the last layer into the labels is done. In [26], it was built from a set of rules *IF* $h_L(x)^i > 0.5$ *THEN* $class_i$, but [26] was concerned only with a binary classification problem where these rules are correct, unlike a multiple classes problem. Another possibility is to build a DT on the activations of the last layer $h_L(X)$ as the input and its classification by the MLP as the output. After initialization, the for loop continues iterating backwards through the layers. At step l, DTs are built for each split node in the DDAG g_{l+1}. Then, the DTs substitute the nodes inside the DDAG, producing a new DDAG g_l with inputs from the lower layer. The substitution of a node by a DT is made by reconnecting input edges to the node into the root of the DT, and then all edges leading into the $True$ and $False$ leaf node are connected to the true and false branches of the initial node, respectively. Finally, the unsatisfiable and redundant nodes are reduced. For such nodes, all samples entering into the node are restricted to meet or not to meet the node's condition, respectively.

Algorithm 1 DeepRED pseudocode using DDAG.

```
1:  function DEEPRED($(h_l(X))_{l=0}^{L}$)
2:      $g_L \leftarrow$ INITIALIZE_DT($h_L(X)$)
3:      for $l = L-1, L-2, \ldots, 0$ do
4:          Set $T$ as the set of unique splits in $g_{h_{l+1}}$
5:          for all $t \in T$ do
6:              $\tilde{X} \leftarrow h_l$
7:              $\tilde{Y} \leftarrow t(h_l(X))$
8:              $DT_t \leftarrow$ BUILD_DT($\tilde{X}, \tilde{Y}$)
9:          end for
10:         $g_l \leftarrow$ SUBSTITUTE($g_{l+1}, (DT_t)_{t \in T}$)
11:         $g_l \leftarrow$ REMOVE_UNSATISFIABLE($g_l$)
12:         $g_l \leftarrow$ REMOVE_REDUNDANT($g_l$)
13:     end for
14:     return $g_0$
15: end function
```

2.2 ANN-DT

The ANN-DT method [21] is one of the simplest methods taking into account only the input-output mapping learned by the network. It builds a DT with axis-parallel splits, but it obtains the input-output pairs for its training in such a way that the NN generates the outputs for their respective inputs.

The main difference from the traditional training method is that more samples are generated if the node's training data contains a low number of samples. It prevents from over-training the DT because its training relies more on the information obtained from the NN model instead of only the training samples. Algorithm 2 summarises the whole recursive process.

Algorithm 2 ANN-DT pseudocode.

```
1:  function BUILD_ANN_DT_NODE(node data $D$)
2:      if $D$ contains less samples then given threshold then
3:          $D \leftarrow D \cup$ GENERATE_SAMPLES($D_{train}$)
4:      end if
5:      $a =$SELECT_ATTRIBUTE($D$)
6:      $t =$SELECT_THRESHOLD($a$, $D$)
7:      Get data $D_{\text{True}}$ and $D_{\text{False}}$ for split $(X_a > t)$
8:      if Stopping rule applies then
9:          return leaf with data $D$
10:     end if
11:     Create node $N$ with split $X_{j_k} > t_k$
12:     $N_{\text{False}} =$BUILD_ANN_DT_NODE($D_{\text{False}}$)
13:     $N_{\text{True}} =$BUILD_ANN_DT_NODE($D_{\text{True}}$)
14:     Set $N_{\text{False}}$ as false branch of $N$
15:     Set $N_{\text{True}}$ as true branch of $N$
16:     return $N$
17: end function
```

A vital component of the ANN-DT algorithm is the attribute selection which should choose the most promising attribute for the next generated split. In [21], the absolute variation is used to this end. A significance of an attribute is measured as the correlation between absolute variation for the considered NN output and that attribute. Unfortunately, the absolute variation is restricted to the one-dimensional output which implies only binary classification problem. For each pair of samples x_i, x_j and a function f monotonic between them the absolute variation is equal to $\left|f(x_j) - f(x_i)\right|$.

The training of DTs chooses splits with the lowest impurity. The impurity is computed as a weighted sum of the same metrics for two sets weighted by the number of samples in the sets. Let us denote D_c the data with classification to the class c and the size of the data $m = \#D$. Instead of commonly used metrics like entropy $-\sum_{c=1}^{C} p_c \cdot \log p_c$, where $p_c = \frac{\#D_c}{m}$, the ANN-DT proposed using the variance $\sum_{i=1}^{m} \left(y_i - \frac{1}{m}\sum_{i=1}^{m} y_i\right)^2$, where y_i is the output of the NN for the sample i. Furthermore, in [6], a gain of fidelity was introduced as a metric for building a DT. It is defined as $1 - \frac{\max_{c \in C}(\#D_c)}{m}$.

The NN defines classification on the whole input space. Because of that, sampling is included, which adds more information about how the NN decomposes the input space into the classes. Therefore, a DT built by the ANN-DT mimic behaviour of the NN on the input space better than a standard DT.

Often, network inputs are assumed to follow a probability distribution of a particular kind, and its parameters are estimated from the training data. It is always necessary to pay attention to the data distribution and sample accordingly. Moreover, sampling for the ANN-DT is specific because samples cannot be arbitrary; conditions on antecedent nodes of a newly created node restrict them. Because of that, the samples need to be tested whether they fulfil the restrictions, and only those that do are accepted.

2.3 HypInv

The NN defines areas of the input space belonging to the same class, and boundaries between these areas are called decision boundaries. A classical DT approximates those boundaries by hyperplanes parallel to axes. On the other hand, the HypInv takes general hyper-planes for the NN approximation [19]. For this purpose, the derivative of the mapping computed by the NN is used.

The authors described this method for binary classification and stated how it could be extended for multiple classes. However, their extension is not generally valid, so we propose other technique at the end of this subsection. Until then, only two classes are considered. Another variant of the original method is that the rules are represented by a DT which is build using only linear splits that were found instead of generating the set of rules directly. For binary classification, the NN will be assumed to compute a function $f : \mathbb{R}^n \to \mathbb{R}^2$, where the outputs are nonnegative, and their sum is one.

The authors originally added oblique splits to rules for each class using conjunction or disjunction. This leads to generating only as many rules as there are classes. However, by adding a new term, the previous structure can be forgotten, e.g., the addition of the disjunction with some subspace leads to forgetting all previous splits in it. Imagine that we have two clusters belonging to the same class. Rules already separate the first cluster. However, the algorithm works locally, so it finds a new split on the decision boundary of the other cluster. If this newly generated split adds a sub-space that contains the whole first cluster, then the information about the first cluster separation is forgotten.

The algorithm works as follows: At first, a point x_1 is initialized. The next step is to find the closest point to x_1 on the decision boundary, denoted as x_0. Then, the hyperplane containing x_0 and perpendicular to the direction $x_1 - x_0$ is created. Finally, the hyper-plane split is added to the set of possible splits on which a DT is built. If the precision of the DT exceeds a predetermined threshold, then it is returned; otherwise the loop continues. In theory, the loop continues until the desired fidelity is achieved, but in practice, the number of loop cycles is restricted.

The whole process is summarised in Algorithm 3. In the following, steps 5, 10, and 12 will be described. In addition, in step 9, building a DT uses all available oblique splits and entropy as an impurity measure.

Algorithm 3 HypInv pseudocode.

```
1:  function HYPINV(neural network $f$, train data $D$, input space $I$)
2:      $x_1 \leftarrow$ INITIALIZE_X1($I$)
3:      $S \leftarrow$ empty set
4:      repeat
5:          $x_0 \leftarrow$ FIND_CLOSEST($f, x_1$)
6:          $n \leftarrow x_1 - x_0$
7:          $\beta \leftarrow n \cdot x_0$
8:          $S \leftarrow S \cup (n \cdot x > \beta)$
9:          $DT \leftarrow$ BUILD_DT($S$, $D$)
10:         $x_1 \leftarrow$ GET_NEXT_X1($DT$, $D$)
11:     until Fidelity of $DT$ on data $D$ excedes threshold
12:     $DT \leftarrow$ REDUCE_ATTRIBUTES($DT$, $D$)
13:     return $DT$
14: end function
```

Closest point on the decision boundary At first, we have to know how is the decision boundary described in the input space. For the binary classification, the answer is simple. It is defined as $\{x \in \mathbb{R}^n \,|\, f(x) = (0.5, 0.5)\}$, provided $\mathbb{R}^n$ is the input space. This set describes a surface in the input space. In addition, every line connecting a point outside that surface and its closest point on the surface is orthogonal to the tangent of the surface at the closest point.

Now, consider the error of the NN for a sample x computed as $E(x) = \frac{1}{2}\|t - f(x)\|^2$, where t is the desired target, which on the decision boundary equals $(0.5, 0.5)$. Below, several ways to obtain the closest point will be introduced.

The first way is to cover the decision boundary by samples evenly by an evolutionary algorithm [19]. The advantage is that this process can be done once at the beginning of the algorithm. Then the closest point is selected from obtained boundary points. However, representing the surface by its points is as good as the density of the points on it. The density decreases exponentially with the dimension of the input space. Hence, it is not a proper method for high dimensional spaces encountered in multimedia data.

The second way uses the inverse to the input-output mapping computed by the feedforward network [19]. Because back-propagation leads to local optima, the inverse, in general, does not give us the closest point on the boundary but just a point on it.

However, a boundary point can be moved along the boundary in the right direction. The idea is based on calculating the direction of a tangent, and the point is iteratively moved in that direction. Sliding along the boundary returns the point $x(t)$ from which the x_1 is in the direction orthogonal to the decision surface. However, it does not have to be a point on the boundary. Therefore after sliding, the inverse mapping is used again for the returned point.

The last way is to modify the error by adding a term representing the distance to x_1, i.e., $E_d(x) = \frac{1}{2}\|t - f(x)\|^2 + \mu\|x_1 - x\|^2$, where μ is a trade-off weight. This approach is forcing x to be simultaneously close to the border by the first term and close to x_1 by the second term. Unfortunately, there is no guarantee that the algorithm of E_d minimisation leads to the global minimum. So, the found point does not need to have either of the desired properties.

The authors tested these approaches on data with input dimensions under 60 without any problem [19]. But inputs with higher dimensions can make an evolutionary algorithm inefficient due to the low density of generated samples covering the decision boundary. So they cannot be used with the multimedia data with hundreds or thousands of features. For those, we found the modified error the most suitable. On the other hand, the process of generating points on the decision boundary is the only way how to deal with non-differentiable NN.

Choosing the next point The next selected point x_1 has to be wrongly classified by current output classifier, i.e., a DT with oblique splits. Originally in [19], two ideas were introduced: The first idea is to take a training sample which is wrongly classified. The second is to generate new samples randomly and to choose the sample that lies the farthest from the boundary among those wrongly classified. In our modification that uses a DT, we propose to find the farthest point from a set of samples attached to the leaf with the largest error.

Attribute reduction So far, the algorithm produced an accurate DT with oblique splits. However, these splits are not much comprehensible because they contain the weighted sum of all attributes. There is no a priory reason to assume that some attribute is missing, i.e., its weight is zero. On the other hand, zeroing small weights may not change the output of the DT. In the original article, authors succeed in testing the attribute reduction with a specified minimal absolute value. However, it is not clear how to choose a proper threshold for weights zeroing.

Multiple classes The main disadvantage of the previous method is that it cannot deal with more than two classes. In binary classification, the above simple condition $f(x) = (0.5, 0.5)$ is fulfilled by each point at the boundary, but in a multi-class case, the situation is more difficult. Imagine an NN with softmax output. A change in the classification of the sample happens when the index corresponding to the maximal output changes. However, it can be any of the other indexes, not just one.

Let us look at the situation when we got a point $x_0 \in \mathbb{R}^n$ from a class $c \in C$ that is wrongly classified. Then we are looking not for any decision boundary but a decision boundary between the class c and all the others because the change of the classes happens on this boundary first. Let us define a function f_{new}, corresponding to the binary classification between class c and the others

$$f^1_{new}(x) = f^c(x),$$
$$f^2_{new}(x) = max_{i \in C'}\left(f^i(x)\right),$$

where $C' = C \setminus \{c\}$. However, function f_{new} is not normalised to sum up to one, so a linear transformation is applied, which results in the final function $f_c = \frac{f_{new}}{f^1_{new}+f^2_{new}}$ that describes two states - the sample is in class c or not. Notice that function f_c is not differentiable due to the maximum in f^2_{new}. On the other hand, this maximum is changing its derivatives only in points where $\exists j, i \in C \setminus \{c\}\left(f^j = f^i\right)$. In addition, the area where this occurs has volume zero in the input space. Hence, a continuous random variable assumes values in this area with the zero probability. Therefore, the gradient can be computed with probability 1.

To sum up, dealing with multiple classes changes the algorithm in one way. The NN output no longer defines a decision boundary, but a new function f_c does, which is described above for any class c. In the algorithm, f_c is always used if the considered x_1 belongs to the class c. Other steps remain unchanged.

3 Application to Multimedia Data

3.1 Data

The data comes from the Week of Science and Technology, a two-week science festival held by the Academy of Sciences of the Czech Republic. Because lectures are popular science, the slides of their presentation contain accompanying text or pictures, and a slide with equations only is rare. For each lecture, a video of the lecture, and the audiovisual footage have been recorded, representing altogether approximately 165 hours of multimedia content. We restricted attention to lectures in the Czech language; these are 124 lectures, each with footage around an hour. Each audio and visual recording is divided according to the sequence of the projected slides. This procedure leads to more training samples, which is desirable. From now on, we denote the content relating to one slide as a document. Each document has its multimedia content which consists of recognised speech from the audio recording, recognized characters on the projected slide by optical character recognition (OCR), and others, which we do not use. A class is assigned to each document according to the scientific field of the whole lecture. By doing so, a noise is added to our data because some lectures deal with the multi-field topic so their slide should be classified differently. In practical use, classification of the lecture will be an induction of its slides classification, e.g., taking the most occurring class in a presentation. Further, we will refer to the text recognized by audio speech recognition as audio.

Table 1: Proposed classes of documents and their frequencies.

Class name	Number of documents
Chemistry	1534
History	644
Biology	3537
Information technology	3573
Physics	3543
Art	391
Social sciences	1222
Geography	1104
Mathematics	294
Other	311

Classes Lectures were divided into thirteen classes according to their field, and four of them were merged into one because they had only a few documents. The classes with their numbers of documents are listed in Table 1.

Data processing Text from the slides was extracted by Tesseract, which has main concepts described in [23]. Google Speech API recognized speech. For more details of processing the original audiovisual input into the textual form, we refer to the master thesis [17] because it is not the main objective of this work.

Text pre-processing The Czech language is a morphologically rich language. So it is convenient to use a stemming algorithm to lower the number of different words occurring in the dataset. Therefore, the recognized audio text was stemmed by a stemming algorithm for the Czech text provided by Petr Chmelar and David Hellebrand from the Faculty of Information Technology, Brno University of Technology [5]. However, the OCR text contains a lot of unrecognized characters, so the stemming deletes many characters, leaving many stemmed words of the OCR text of the document empty.

Moreover, in the OCR text, there are many recognized numbers, but they are often used as particular information linked to the topic of the lecture, e.g., as altitude, era, or as indexes. There are three main ways how to deal with numbers. Firstly, do not process numbers at all. Secondly, delete words containing numbers as was done in [17]. The third option is to substitute the number with a unique character as was done in [10]. We prefer the third option because no valid information about the field will be lost.

Even though data consists of text, it has to be converted into a numeric input on which the NN can be trained. For this purpose, several methods are available. One of them is the substitution of words by the vectors, so-called words embedding. The state-of-the-art embedding is obtained by the sub-word information skip-gram model [2]. However, this approach requires the use of convolutional neural networks, which are not supported by implemented methods.

Therefore, we chose to use another conversion, described below.

Bag of words The method bag of words (BOW) uses a dictionary of all words comprised in data. The document is represented by a vector with a length of the size of the dictionary containing integer values representing frequencies of words that occur in the document. So we lose information about word order by this procedure.

Moreover, this method has two more drawbacks. One drawback is that the dictionary of all used words could be large. This drawback can be partly solved by substituting words with low frequency by a unique word or by some other dimension reduction. The second drawback arises with documents with different length where one word could have lower frequency just because of the length of the document. This issue could be solved by applying weights instead of frequencies. Aside from the fact that frequencies depend on document length, there are words in all languages that are very frequent but do not carry any information, e.g., articles in English. So there is also a reason to decrease the weight of frequent words.

A method called term frequency - inverse document frequency (TF-IDF) has been developed to solve issues with document length dependence. The article [20] offers many ways how to weight words in the BOW method. In the following, the variant used in [17] will be described. Let us denote the frequency of the word τ in the document δ as $c_{\delta,\tau}$, then the TF-IDF value of this word τ in document δ denoted as $h_{\delta,\tau}$ is computed by

$$h_{\delta,\tau} = \frac{c_{\delta,\tau}}{max_{\tau}\left(c_{\delta,\tau}\right)} \cdot \log\left(\frac{d}{d_{\tau}}\right), \tag{1}$$

where d is a total number of documents, d_{τ} is the total number of documents which contains at least one term τ. The TF-IDF value is higher if the relationship between term τ and document δ is stronger. Finally, stemmed audio text converted into TF-IDF form will be used for the lecture classification. However, the dictionary for the OCR text is too large, and the vector representations of documents are sparse. Therefore, for further processing of OCR data, we use the approach described below.

N-gram In natural language processing (NLP), an N-gram usually means sequences of N words [4]. A BOW model of these sequences is then produced. This is a way how to incorporate the meaning hidden in the context.

However, there is another way to see the N-gram model the character N-gram. It is considering sequences of characters instead of sequences of words. This approach is recommended in [4] for OCR text since OCR text is generated character by character. Because the N-gram model is very close to BOW model, the same method to shift from frequencies to weights is applied here, too. Therefore, the N-gram input derived from the OCR text transformed by TF-IDF will be used as OCR input data.

Dimensionality reduction The inputs described above, namely BOW audio data and joined character N-grams with $N \in \{1,2,3\}$ of OCR data, have very high dimensionality: 34910 and 20586, respectively. For them, we have 15412 and 16965 non-empty samples. Because of the disproportion of the dimensionality and number of samples, a dimension reduction procedure is appropriate. However, standard methods like principal component analysis transform the coordinate system, and rules in the new coordinate system are not comprehensible. So we focus on methods that select the best attributes instead, through leaving out rarely occurring words, more precisely, through replacing all of them with a unique word.

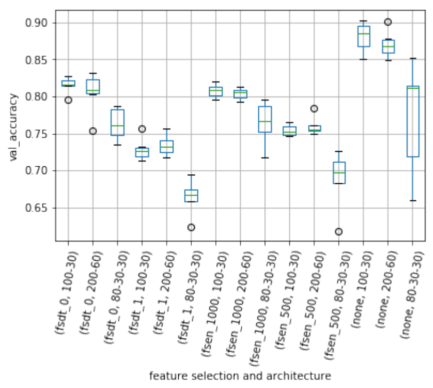
On the other hand, building a DT or more precisely ANN-DT also provides some measures to compare attributes and their possible contribution to the classification. For example, one of these measures is entropy, which we use to choose the attributes that provide the split with the highest entropy decrease. This dataset will be denoted as *fsen_500* and *fsen_1000,* where 500 and 1000 best attributes were selected, respectively. In addition, the dataset with all available attributes will be denoted as *all.* Also, the built DT with multiple-class training data provides attributes that are needed to perform its classification. Therefore, we have taken attributes occurring in the DT as the second feature selection method. Corresponding datasets will be denoted as *fsdt_0* and *fsdt_1* where the indexes that occur not at all and at most once in the DT were deleted. It leads to the dimensions under 1500 and 500, respectively, for both audio and OCR datasets.

3.2 Experiment Design

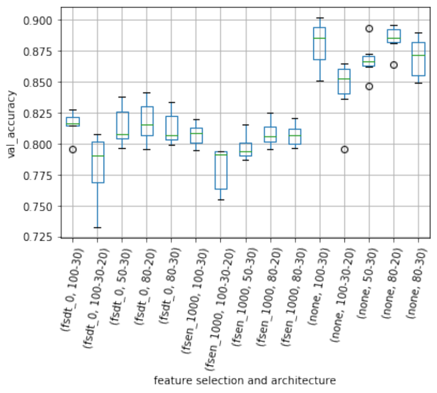
The rule extraction process is separated from NN training to mimic a real word problem where a NN is trained at first. Only afterwards, if it has satisfactory performance, then the rules are extracted. The employed methods are focused on the MLP, so the experiments will be done only on those architectures. The architectures will be described by the number of neurons in the hidden layers separated by a dash.

As was mentioned in the previous section, we used a TF-IDF representation of stemmed text obtained from the audio recording denoted as audio. Apart from that, the TF-IDF representations of 1-, 2- and 3-grams of OCR data were joined into one dataset. NNs were trained on one dataset without reduced dimension (*all*) and four datasets with reduced dimension based on index occurrence in DT (*fsdt_0* and *fsdt_1*) or entropy ordering (*fsen_500* and *fsen_1000*). Since each of those combinations is considered both for the OCR data and for the audio data, there are altogether 10 datasets.

The first experiment conducted on each of the ten datasets has used NNs with architectures $100-30$, $200-60$, and $80-30-30$. Figures 1a and 2a show the results of the NNs for audio and OCR datasets, respectively. The dimension reduction on the audio data seems to reduce the

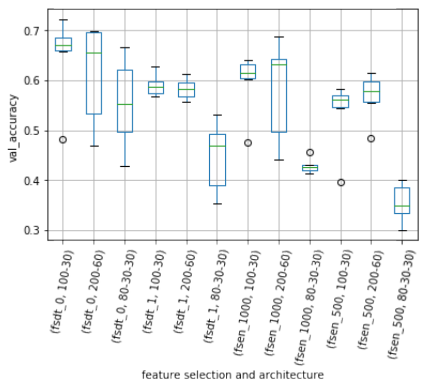


(a) The first experiment with all datasets.

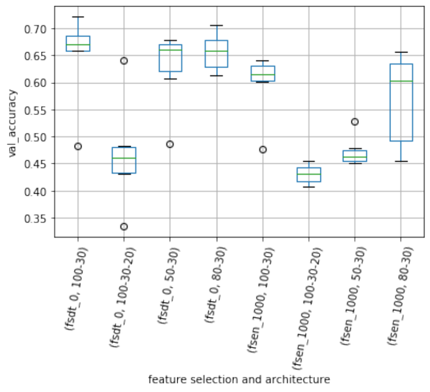


(b) The second experiment exploring the architectures similar to the $100-30$ on datasets with satisfactory performance.

Figure 1: Performances of the trained NNs on audio data along with the data with reduced dimensionality.



(a) The first experiment with all datasets.



(b) The second experiment exploring the architectures similar to the $100-30$ on datasets with satisfactory performance.

Figure 2: Performances of the trained NNs on OCR data along with data with reduced dimensionality.

performance of the NN substantially. However, even with those data, the best NN achieves around 80% of validation accuracy, which we also considered as a good result. The situation is different for the OCR data, where dimension reduction helps the classifier to increase its performance. The OCR results are shown in Figures 2. Looking at the difference between NN architectures, it seems that architectures $100-30$ and $200-60$ are pretty much comparable for audio data and the architecture $200-60$ has a higher variance in the OCR domain. A second experiment was conducted to explore a parameter space around the architecture $100-30$, and it is shown in Figures 1b and 2b for audio and OCR datasets, respectively.

The final selected architecture for the audio data was chosen $80-20$ for input without feature selection even though the architecture $100-30$ has a similar performance. For the audio data with feature selection *fsdt_0* and *fsen_1000*, the same architecture $100-30$ was chosen. In the OCR domain, we decided to use only the data *fsdt_0* with the architecture $80-30$, which does not have any classification accuracy under 50%. Other datasets will not be considered due to their low NN classification accuracy.

3.3 Results and Their Discussion

The experiments will be divided into narrowed binary classification and full multiclass classification. The binary classification differentiates most often occurring class (*information technology*) against the others, and the same number samples as the samples from the most often occurring class is randomly selected. Only the data on which the corresponding best NN has a good performance will be considered, namely: *all*, *fsdt_0*, *fsen_1000* for audio data, and only *fsdt_0* for OCR data. The rule extraction method will be applied with several settings.

In particular:

- Attribute selection using absolute variation combined with variance as impurity measure denoted as *absvar-var*, and only impurities entropy, a gain of fidelity, and variation without attribute selection denoted as *entropy*, *fidgain*, and *var*.
- For DeepRED, we chose always to build an initial DT and experiment with the activations omitted before the softmax layer.

- For HypInv, we emphasize two numbers which represent the number of generated splitting hyper-planes and the maximal depth of the tree, respectively.

In addition, we want to measure the comprehensibility of the extracted rules. For this purpose, we selected four values: The number of rules, the number of attributes used by the whole set of rules, the mean rule length, and the mean number of attributes occurring in each rule. The number of attributes occurring in the rule describes to what extent rules are using each attribute, e.g., axis-parallel rules have at most the lower and upper bound for each attribute, i.e., the number of attributes is half in comparison with rule length.

Also, the fidelity, fidelity computed on validation data, and validation accuracy are computed. All of those values are presented in the combination of their mean and standard deviation except for the number of rules in the case of DeepRED where the minimum and the maximum is indicated.

The results of the experiments described above are shown in Table 2. Friedman statistical test with Nemenyi post-hoc test on the significance level 5% reveals which of the found differences between methods and their settings are significant. Below, these differences are described.

Firstly, from the view of fidelity, significantly worse than all the other methods appear to be DeepRED and the *absvar-var* variant of ANN-DT. On the other hand, Hypinv and ANN-DT with *entropy* and *var* as parameters were pretty good and no difference was revealed between them. The validation fidelity and accuracy lead to the same conclusions. HypInv was recognized as a method generating the significantly shortest rules with the most often occurring features among employed methods. Moreover, DeepRED appears to have many features per rule, too. DeepRED also has a significantly higher length of the rules than all the other methods.

Furthermore, Table 2 shows that methods that using axis-parallel splits fail to have a good performance for all audio data. Because those methods included the ANN-DT, which is almost identical to the conventional DT, we assume that decision trees have low accuracy on given datasets. The inefficiency of the DT also appears in the last step of the DeepRED method. Table 3 provides the fidelities of the DDAG constructed by the DeepRED before and after the last substitution was made. It shows that the error of the DeepRED algorithm stems from the last step where DTs are built on the input data. Also, Figure 3 confirms this statement by showing that the distribution of the accuracy is in the last step shifted to lower values compared to the last but one step.

Moreover, some of the methods can be applied to the whole ten class datasets. Table 4 shows results for such methods, i.e., ANN-DT and HypInv. Settings *absvar-var* and *var* were omitted because they can be only applied to binary classification. The DeepRED is not covered because experiments had shown that its performance is very low for multiple classes classification, and its computation time requirement is enormous. The behaviour of the ANN-DT method is similar to binarized data because it performed well on the OCR data. However, the HypInv method achieved validation fidelity significantly better than all other methods, which we believe caused by its ability to find more complex splits than axis-parallel, and those splits describe data more precisely. In addition, for the audio data without feature selection, HypInv exceeds the predetermined fidelity threshold of 95% in each observation. However, it leads to the largest difference in fidelity between training and validation.

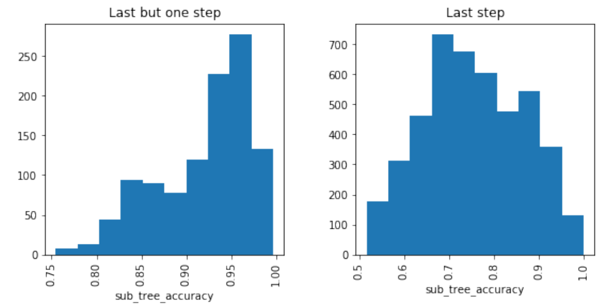


Figure 3: Histograms for sub-trees built in the DeepRED in the last but one and the last step.

In addition, Table 5 shows an example of extracted rules for each of the employed methods from the binary class audio data without feature selection. The most relevant feature in DeepRED is with subscription *pocitact* because it is a world that is almost the same as *počítač* - the computer in the Czech language. Moreover, the rule of the HypInv is inhibited by the word *earth*, which also looks promising. However, other words cannot be simple revealed why they occur. For example, the world *bunout* with no meaning in the Czech language maybe there just because it provides some random class separation.

4 Conclusion

The available methods for rule extraction from trained neural networks provide a wide range of possibilities. We chose to implement two methods taking into account only the input-output mapping learned by the network, i.e., ANN-DT and HypInv that deal with any NN. Also, a method called DeepRED was implemented. Experiments have shown that these methods differ in their performance on different datasets. In addition, for methods that are using DT, i.e., ANN-DT and DeepRED, it is crucial to have data on which DT achieves satisfactory results. The most consistent performance was observed for the HypInv method. However, its expressive power is generally not as good as that of other implemented methods. On the other hand, DeepRED results were the least consistent because it generated DDAG with random depths which leads to many long rules in some cases.

Table 2: Comparison of the implemented methods on the binarized datasets. Averages and standard deviations are computed by 10-fold cross-validation. Cells in the column rule_count where the number of rules is marked with * are intervals in which the value occurs. Columns denote fidelity (*fidelity*), validation accuracy (*val_accuracy*), validation fidelity (*val_fidelity*), the number of rules (*rule_count*), the number of attributes used by the whole set of rules (*index_count*), the mean rule length (*rule_len*), and the mean number of attributes occurring in each rule (*rule_index*), respectively.

Data	Method	Settings	fidelity (%)	val_fidelity (%)	val_accuracy (%)	rule_count	index_count	rule_len	rule_index
audio_stem_all fsdt_0	ANN-DT	absvar-var	53.18 ± 0.42	53.14 ± 1.93	50.11 ± 2.09	14.9 ± 27	12.9 ± 25	2.06 ± 3.43	2.06 ± 3.4
		entropy	57.23 ± 1.38	56.78 ± 2.13	54.93 ± 1.59	10.6 ± 6.3	9.40 ± 6.2	4.05 ± 1.72	4.01 ± 1.7
		fidgain	58.31 ± 1.31	57.07 ± 2.96	54.33 ± 2.76	26.5 ± 4.7	22.8 ± 4.7	5.80 ± 0.36	5.76 ± 0.39
		var	57.01 ± 1.26	56.37 ± 2.14	54.71 ± 2.02	11.0 ± 4.2	9.40 ± 3.6	3.94 ± 1.10	3.94 ± 1.1
	DeepRED	build_first	54.18 ± 1.87	53.25 ± 3.58	52.55 ± 3.03	2 - 503*	7.80 ± 6.1	4.90 ± 3.61	4.17 ± 2.7
		del_last & build_fist	52.86 ± 2.62	53.34 ± 3.48	53.37 ± 3.03	57 - 6060*	22.7 ± 8.1	12.01 ± 2.87	10.7 ± 2.4
	HypInv	20-6	80.39 ± 1.86	80.76 ± 3.00	77.07 ± 2.53	7.30 ± 3.6	1533 ± 1.9	3.44 ± 0.95	1533 ± 1.9
audio_stem_all fsen_1000	ANN-DT	absvar-var	52.01 ± 0.59	51.92 ± 1.88	50.11 ± 1.70	17.3 ± 22	15.3 ± 21	2.99 ± 3.86	2.98 ± 3.9
		entropy	58.55 ± 1.30	57.93 ± 1.77	55.25 ± 1.77	6.50 ± 2.4	5.40 ± 2.3	3.06 ± 0.62	3.06 ± 0.62
		fidgain	56.02 ± 1.23	54.78 ± 2.04	53.05 ± 1.57	24.7 ± 7.6	21.8 ± 6.2	5.82 ± 0.54	5.72 ± 0.52
		var	58.83 ± 1.10	58.12 ± 1.78	55.41 ± 1.96	11.3 ± 5.7	10.1 ± 5.5	4.49 ± 1.68	4.49 ± 1.7
	DeepRED	build_first	52.05 ± 3.56	51.49 ± 3.16	52.38 ± 2.04	9 - 1361*	14.7 ± 8.0	8.50 ± 4.24	7.38 ± 3.4
		del_last & build_fist	49.78 ± 1.65	49.93 ± 1.14	51.76 ± 1.26	2 - 13278*	23.9 ± 11	12.44 ± 5.00	10.6 ± 4.2
	HypInv	20-6	77.74 ± 2.54	77.76 ± 2.61	73.14 ± 1.82	11.6 ± 6.5	999 ± 0.3	3.90 ± 1.22	999 ± 0.32
audio_stem_all	ANN-DT	absvar-var	51.15 ± 0.93	50.87 ± 2.70	49.71 ± 2.16	2.20 ± 2.7	1.20 ± 2.7	0.74 ± 1.60	0.74 ± 1.6
		entropy	59.78 ± 0.90	59.59 ± 2.63	55.84 ± 2.07	6.60 ± 2.9	5.40 ± 2.8	3.37 ± 1.28	3.33 ± 1.3
		fidgain	58.42 ± 2.90	56.63 ± 3.64	55.73 ± 3.97	27.5 ± 11	24.2 ± 9.7	5.75 ± 0.92	5.71 ± 0.87
		var	59.82 ± 0.88	59.69 ± 2.72	56.01 ± 2.21	8.10 ± 5.6	7.00 ± 5.3	3.45 ± 1.51	3.45 ± 1.5
	DeepRED	build_first	51.54 ± 1.95	52.16 ± 2.68	51.93 ± 3.13	3 - 75*	7.60 ± 3.1	4.95 ± 1.81	4.37 ± 1.5
		del_last & build_fist	51.89 ± 2.01	51.66 ± 3.06	51.17 ± 2.48	3 - 1019*	11.8 ± 9.9	6.67 ± 4.23	5.95 ± 3.8
	HypInv	20-6	88.28 ± 5.45	87.17 ± 5.99	83.37 ± 3.84	3.00 ± 2.1	24437 ± 161	1.57 ± 1.20	24437 ± 161
ocr fsdt_0	ANN-DT	absvar-var	59.94 ± 4.12	59.20 ± 4.65	54.32 ± 4.43	46.2 ± 30	42.0 ± 27	6.45 ± 2.56	6.38 ± 2.5
		entropy	81.17 ± 2.02	78.02 ± 1.12	70.88 ± 2.61	105 ± 26	86.4 ± 19	8.31 ± 0.35	8.17 ± 0.41
		fidgain	79.75 ± 3.01	75.74 ± 1.84	70.38 ± 5.77	118 ± 48	97.4 ± 37	7.63 ± 1.17	7.50 ± 1.1
		var	70.65 ± 6.96	69.68 ± 6.56	66.52 ± 4.88	49.7 ± 25	45.9 ± 24	6.93 ± 2.29	6.91 ± 2.3
	DeepRED	build_first	62.36 ± 7.09	62.03 ± 6.32	60.18 ± 3.95	108 - 50763576*	37.7 ± 11	25.55 ± 8.94	18.0 ± 5.4
		del_last & build_fist	65.70 ± 5.08	65.65 ± 5.63	61.62 ± 4.91	82 - 3324312*	34.2 ± 16	17.80 ± 6.88	13.2 ± 4.1
	HypInv	20-6	73.14 ± 3.11	72.91 ± 3.68	69.09 ± 7.35	19.8 ± 5.2	711 ± 213	4.87 ± 0.22	711 ± 213

Table 3: The change of DDAG fidelity in the last step of the DeepRED method for all considered binarized datasets.

Data	Settings	Fidelity before last step (%)	Fidelity after last step (%)
audio_stem_all fsdt_0	**build_first**	85.03 ± 0.01	54.18 ± 0.02
	del_last & build_fist	85.80 ± 0.02	52.86 ± 0.03
audio_stem_all fsen_1000	**build_first**	78.91 ± 0.07	52.05 ± 0.04
	del_last & build_fist	82.42 ± 0.02	49.78 ± 0.02
audio_stem_all	**build_first**	97.49 ± 0.01	51.54 ± 0.02
	del_last & build_fist	98.53 ± 0.01	51.89 ± 0.02
ocr fsdt_0	**build_first**	83.21 ± 0.18	62.36 ± 0.07
	del_last & build_fist	94.76 ± 0.01	65.70 ± 0.05

We do not present an analysis of the extracted rules, because we want to keep our process as general as possible. Also, comprehensibility of the rules is not proven by their thorough discussion but just by possible measures of comprehensibility which support statistical analysis of their properties. So the experimental process can be repeated with almost all kinds of data also with the final comparison of the methods on those data.

Due to the reproducibility of the process, it can be the basis of choosing the right rule extraction algorithm for specific data in the future, or even, it can be a part of an automated process that receives data or NN and returns rules extracted by the most suitable algorithm.

Acknowledgement

The research reported in this paper has been supported by the Czech Science Foundation (GAČR) grant 18-18080S.

References

[1] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[3] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Classification and regression trees. wadsworth int. *Group*, 37(15):237–251, 1984.

[4] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[5] Petr Chmelař, David Hellebrand, Michal Hrušecký, and Vladimír Bartík. Nalezení slovních kořenů v češtině. In *Znalosti 2011: Sborník příspěvků 10. ročníku konference*, pages 66–77. VŠB-Technical University of Ostrava, 2011.

[6] Mark W Craven. Extracting comprehensible models from trained neural networks. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1996.

[7] Joachim Diederich, Alan B Tickle, and Shlomo Geva. Quo vadis? reliable and practical rule extraction from neural networks. In *Advances in Machine Learning I*, pages 479–490. Springer, 2010.

[8] Wlodzislaw Duch, Rafal Adamczak, and Krzysztof Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, 12(2):277–306, 2001.

[9] Artur SD'Avila Garcez, Luis C Lamb, and Dov M Gabbay. *Neural-symbolic cognitive reasoning*. Springer Science &

Table 4: Comparison of the implemented methods on datasets with ten classes. Averages and standard deviations are computed by 10-fold cross-validation. Columns denote fidelity (*fidelity*), validation accuracy (*val_accuracy*), validation fidelity (*val_fidelity*), the number of rules (*rule_count*), the number of attributes used by the whole set of rules (*index_count*), the mean rule length (*rule_len*), and the mean number of attributes occurring in each rule (*rule_index*), respectively.

Data	Method	Settings	fidelity (%)	val_fidelity (%)	val_accuracy (%)	rule_count	index_count	rule_len	rule_index
audio_stem_all fsdt_0	ANN-DT	entropy	28.22 ± 0.83	29.23 ± 1.40	27.59 ± 1.10	14 ± 3.4	12.5 ± 2.7	4.49 ± 0.57	4.21 ± 0.44
		fidgain	31.45 ± 0.44	31.67 ± 1.09	29.76 ± 1.06	31.2 ± 4.7	26.5 ± 3.6	5.74 ± 0.44	5.71 ± 0.44
	HypINV	30-10	72.72 ± 2.41	70.86 ± 2.15	64.33 ± 1.74	184 ± 24	1571 ± 1.8	8.53 ± 0.12	1571 ± 1.8
audio_stem_all fsen_1000	ANN-DT	entropy	29.17 ± 1.03	29.40 ± 0.80	27.37 ± 0.89	17.3 ± 3.9	14.9 ± 3.7	4.88 ± 0.61	4.57 ± 0.76
		fidgain	32.56 ± 0.51	32.11 ± 0.81	29.93 ± 1.04	29.1 ± 12	24.9 ± 9.1	5.34 ± 0.80	5.29 ± 0.80
	HypINV	30-10	69.35 ± 2.10	68.47 ± 2.39	61.80 ± 2.13	233 ± 29	1000 ± 0.0	8.77 ± 0.10	1000 ± 0.0
audio_stem_all	ANN-DT	entropy	27.78 ± 0.86	27.72 ± 1.12	27.45 ± 1.30	13.4 ± 5.6	11.6 ± 4.6	4.25 ± 0.83	4.06 ± 0.58
		fidgain	31.21 ± 0.25	30.05 ± 1.10	29.80 ± 1.08	31.8 ± 6.8	27.6 ± 4.9	5.69 ± 0.69	5.66 ± 0.63
	HypINV	30-10	95.51 ± 0.41	88.78 ± 1.10	84.76 ± 1.07	136 ± 18	33317 ± 174	8.11 ± 0.18	33317 ± 174
ocr fsdt_0	ANN-DT	entropy	49.56 ± 4.75	48.44 ± 4.86	35.73 ± 3.66	15.4 ± 4.6	13.3 ± 3.5	4.41 ± 0.47	4.36 ± 0.47
		fidgain	58.41 ± 2.20	56.48 ± 2.20	43.74 ± 5.91	33.7 ± 12	30.8 ± 11	5.77 ± 0.61	5.71 ± 0.65
	HypINV	30-10	87.52 ± 3.58	86.83 ± 4.19	61.88 ± 5.99	146 ± 26	1210 ± 0.0	8.22 ± 0.26	1210 ± 0.0

Table 5: Examples of extracted rules by employed methods on the binary classification audio dataset without feature selection. A subscript denotes the stemmed word to which the TF-IDF value belongs. The HypInv rule contains a linear combination of almost all attributes, so only the four largest in absolute value are shown.

Method	Example of extracted rules
ANN-DT	IF $(x_{\text{bunout}} \leq 0.04)$ *and* $(x_{\text{mozik}} \leq 0.14)$ *and* $(x_{\text{mozk}} > 0.13)$ THEN *information technology*
DeepRED	IF $(x_{\text{pocitact}} > 0.56)$ *and* $(x_{\text{potom}} \leq 0.27)$ *and* $(x_{\text{bunout}} > 0.04)$ THEN *information technology*
HypInv	IF $(\ldots - 2.0 \cdot x_{\text{prosit}} + \ldots + 1.5 \cdot x_{\text{earth}} + \ldots - 1.3 \cdot x_{\text{caj}} + \ldots - 1.4 \cdot x_{\text{srozum}} \leq 1)$ THEN *information technology*

Business Media, 2008.

[10] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4. sn, 2006.

[11] David Hand, Joost N Kok, and Michael R Berthold. *Advances in Intelligent Data Analysis: Third International Symposium, IDA-99 Amsterdam, The Netherlands, August 9-11, 1999 Proceedings*. Springer Science & Business Media, 1999.

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

[13] Kurt Hornik, Maxwell Stinchcombe, Halbert White, and Peter Auer. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. *Neural Computation*, 6(6):1262–1275, 1994.

[14] Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural networks*, 5(3):501–506, 1992.

[15] Sushmita Mitra and Yoichi Hayashi. Neuro-fuzzy rule generation: survey in soft computing framework. *IEEE transactions on neural networks*, 11(3):748–768, 2000.

[16] Sushmita Mitra and Yoichi Hayashi. Neuro-fuzzy rule generation: survey in soft computing framework. *IEEE transactions on neural networks*, 11(3):748–768, 2000.

[17] Petr Pulc. Research into approaches to the classification of audiovisual recordings of lectures and conferences. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Department of Software Engeneering, 2014.

[18] J Ross Quinlan. C4. 5: Programming for machine learning. *Morgan Kauffmann*, 38:48, 1993.

[19] Emad W Saad and Donald C Wunsch II. Neural network explanation using inversion. *Neural networks*, 20(1):78–93, 2007.

[20] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[21] Gregor PJ Schmitz, Chris Aldrich, and Francois S Gouws. Ann-dt: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6):1392–1401, 1999.

[22] Roberta Siciliano and Francesco Mola. Multivariate data analysis and modeling through classification and regression trees. *Computational Statistics & Data Analysis*, 32(3-4):285–301, 2000.

[23] Ray Smith. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[24] Alan B Tickle, Robert Andrews, Mostefa Golea, and Joachim Diederich. The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–1068, 1998.

[25] Jan Zilke. Extracting rules from deep neural networks. Master's thesis, Technische Universität Darmstadt, 2015.

[26] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. Deepred–rule extraction from deep neural networks. In *International Conference on Discovery Science*, pages 457–473. Springer, 2016.