Reconstruction of Boolean Functions from Deep Neural Networks

Bachelor-Thesis von Camila González aus Buenos Aires
Tag der Einreichung:

1. Gutachten: Prof. Dr.-Ing. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Reconstruction of Boolean Functions from Deep Neural Networks

Vorgelegte Bachelor-Thesis von Camila González aus Buenos Aires

1. Gutachten: Prof. Dr.-Ing. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 28th April 2017

(C. González)

Abstract

Due to improvements in the training practices, the popularity of deep neural networks and their applicability to a wider set of problems have increased considerably in the last few years. This has brought about a renewed interest in increasing their interpretability, an aspect where they fall behind other prediction models.

It is unclear what provides a sufficiently expressive explanation of the inner workings of a deep model, but symbolic representations in the form of rule sets have been proven capable of illustrating their behaviour as a whole, as well as the hidden concepts they model in the intermediate layers.

Diverse methods have been developed for this purpose. However, most approaches are not scalable to deep architectures with a high number of hidden units and which were trained with huge amounts of data. In order to enable rule extraction in such situations, the search space of possible concepts has to be appropriately reduced.

Within this thesis, the possibility is explored of extending techniques which have been shown to facilitate the extraction of rule representations from shallow neural networks to deep architectures. These methods, which go from altering the internal structure of the network to choosing appropriate values to partition the activation ranges, are tested in combination with an existing algorithm for extracting rules from deep models.

Instead of using datasets which combine the attributes in an unclear manner, the networks are trained to reproduce predefined boolean concepts so it can later be assessed to what degree the patterns were captured in the rule sets.

The evaluation shows that reducing the connectivity of the neural networks significantly assists latter rule extraction, as does encouraging minimal or maximal activation states. It is also demonstrated that these can be discretised without causing a significant decrease in accuracy.

Zusammenfassung

Aufgrund von Verbesserungen in Trainingsverfahren ist die Bedeutung von Tiefen Neuronalen Netzen und die Anwendbarkeit dieser auf eine breitere Menge von Problemen in den letzten Jahren gestiegen. Somit ist es vorteilhaft, ein tieferes Verständnis der Vorgänge innerhalb der Modelle zu entwickeln.

Während bisher noch unklar ist, welche Form eine anschauliche Erklärung der inneren Abläufe besitzen kann, haben sich symbolische Darstellungen in der Form von Regelmengen als geeignet erwiesen, das Verhalten solcher Modelle sowohl als ganzes und als auch innerhalb der Zwischenschichten wieder zu geben.

Zwar wurden verschiedene Methoden zu diesem Zwecke konzipiert, jedoch sind die meisten Ansätze nicht mit sehr tiefen Modellen und großen Mengen von Daten skalierbar. Um dennoch Regelmengen erkennen zu können, ist es notwendig, den Suchraum der Möglichkeiten entsprechend zu reduzieren.

Im Rahmen dieser Arbeit wird die Möglichkeit einer Übertragung von Ansätzen, welche die Regelerkennung in flachen neuronalen Netzen vereinfachen, auf tiefe Netze untersucht. Diese Methoden, welche von Änderungen der inneren Struktur der Netze bis hin zu geeigneter Aufteilung der Aktivierungsreichweite reichen, werden in Kombination mit einem bereits existierenden Algorithmus zur Regelerkennung in Tiefen Neuronalen Netzen getestet.

Die Netzwerke werden anstatt mit Datensätzen, welche Attribute in einer unbekannten Weise kombinieren, mit künstlichen trainiert, um vordefinierte Boolesche Ausdrücke zu modellieren. Dies ermöglicht es bei späterer Betrachtung festzustellen, inwiefern die logischen Ausdrücke auch noch in den Regelmengen zu finden sind.

Die Auswertung der Daten zeigt, dass eine Reduzierung der Verknüpfungen innerhalb eines Netzes zu einer besseren Regelerkennung beiträgt, ebenso wie die Förderung von maximalen oder minimalen Erregungszuständen. Außerdem wird gezeigt, dass eine Diskretisierung dieser durchgeführt werden kann, ohne die Genauigkeit der Ergebnisse bedeutend zu verringern.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

Deep neural networks achieve state of the art performance in an array of different fields, like computer vision, speech recognition and machine translation. They can be leveraged both in supervised and unsupervised problem formulations, as they automatically learn insightful features out of unprocessed data. In the last few years, they have considerably risen in popularity as advancements in the training practices and availability of user friendly frameworks have made it much simpler to train accurate models, as long as sufficient data is available.

However, the fact that the models are governed by a considerable number of parameters makes tracing the path that led to a classification an arduous process, which is why they are ofter regarded as *black box* models. This is a significant shortcoming, as it makes them unsuitable for safety critical applications and domains where there are juridical barriers which either explicitly forbide their use or implicitly discourage it by making the user liable for the model's decisions. Therefore, in fields such as health care and criminal sentencing, comprehensible models like decision lists or trees are favored because they provide understandable evidence to support their predictions (Liu and Li, 2005; Malioutov and Varshney, 2013).

As more encouraging results are obtained from testing the technology within these fields, the lack of interpretability is proving to be a significant obstacle in infiltrating them. Last year, convolutional neural networks were used in the autonomous driving approach outlined in Bojarski et al. (2016) which, in contrast to other systems, relies solely on image data processed by the networks and is not explicitly trained to recognize road features. Deep autoencoders have, as well, been used to find feature representations of electronic health records, which later proved to be highly effective in predicting an array of diseases (Miotto et al., 2016).

Another issue is that in domains where decisions have critical consequences, 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 the recommendation. 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 is a relevant criteria for eliciting trust (Kayande et al., 2009).

Concerning the issue of liability, in Burri (2016) it is stated that human actors are only held accountable for behaviour committed purposely, and in areas such as driving where people are also permitted some margin of error, this matter could be addressed by altered insurance policies, as well as by regularizing the certification processes. However, although a reallocation of responsibility could make the user less liable, he or she will likely be accountable for deciding to use the technology in a specific setting.

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 penalise a certain bias when manually making decisions, but this may not be explicitly included in the objective the model optimises. 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 data should be excluded from prediction models even if it was legally obtained, such as that which shows 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.

Apart from these juridical concerns, another argument in favor of increasing interpretability is that insight into the feature combinations generated by deep models which achieve particularly low error rates could be leveraged to improve other models, enabling a deliberate selection of meaningful features.

Compared to neural networks, *if-then* rule sets are regarded as being much easier to understand. This is partly because they provide a symbolic representation which more closely resembles the way humans model logic. Also, each rule can be observed individually, so only a limited amount of information must be considered at any time. This advantage sometimes makes them preferable over decision trees, as well as the fact that they can be more flexibly pruned (Freitas, 2013).

The goal of this work is to obtain both an accurate model which explains the behaviour of the deep neural network as a whole, as well as a compact and expressive description of the logic that takes place within the network.

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

Problem Statement

Within this work, deep neural networks are trained to model predefined logical patterns. The objective is to regain these constructs in the form of rule representations. The disjunctive normal form of the functions is considered, as any expression can be represented in this manner and this is the typical syntax for sets of decision rules which make up symbolic machine learning models.

Besides reconstructing the boolean functions from the trained networks, this should be done in such a manner that the internal behaviour is left exposed and is sufficiently expressive. The expectation is that by altering the neural networks in certain ways and setting adequate constraints on the search space of possible rules, semantically rich intermediate concepts can be retrieved.

Main Contibutions and Findings

Ample research has been devoted to extracting rule representations from neural networks. Some of this aims to illustrate not only the predictive behaviour but also the function of the individual hidden units, which brings about an additional set of advantages. Yet, most of the methods enforce considerable constraints on the topology of the networks and the form of the attributes, which makes then unsuitable for the much larger architectures that have recently risen in popularity.

Alternatively, treating each two consequent layers as a separate classification problem and merging the partial models has been shown to be a viable way to construct such representations while leaving a trace of the hidden logic. However, the intermediate concepts can become highly complex, and the rule sets do not paint a clear picture of the function of each neuron.

Intuitively, each neuron that makes up an artificial neural network reaches an *excited* state if the added contributions of its incoming connections is sufficiently strong. However, in practice neurons can output any value within a continuous range, which makes describing their effect on the subsequent layer in a compact manner substantially more difficult.

Within this work, two methodologies are presented for altering a trained neural network so its internal structure is simpler to describe. The first consists on reducing the network connectivity so the state of each neuron only depends on a reduced number of units from the previous layer, and the second looks to polarise the activations so the neurons are either minimally of maximally active. Additionally, two different discretisation techniques are explored that select a reduced number of thresholds to divide the range of each hidden unit. Finally, a post-pruning method is introduced.

The effect of these techniques on an existing algorithm for extracting rule representations from deep neural networks is evaluated. The results show that reducing network connectivity is an effective way to simplify the inner working of the network so they can be explained more easily, and that when this is combined with polarisation of the activations a minimal number of thresholds per neuron is sufficient to explain its main function. Also, it is shown that very accurate models can be extracted if only a small amount of predefined split values are selected for explaining each layer, which makes extracting rule representation viable for very large architectures.

Thesis Structure

This work is structured as follows. Chapter 2 summarises the fundamentals of classification problems and neural networks which are required to understand the subsequent chapters. Chapter 3 gives an overview on related work in the fields of network pruning and discretisation approaches, as well as on existing techniques for extracting rule representations from neural networks. In Chapter 4, the methodology is portrayed. In Chapter 5 the experimental setup is described, and the selected boolean formulas that are to be reconstructed are presented. Chapter 6 illustrates the obtained results. Chapter 7 outlines some areas of future and existing research that would complement the approach. Finally, Chapter 8 concludes the work with a synopsis of the relevant takeaways.

Terminology

A *literal* or *condition* signifies whether an input or activation value of a hidden unit for an observation exceeds a threshold, and takes the form $h_{l,n} > t$ or $h_{l,n} <= t$ for hidden units and $x_n > t$ or $x_n <= t$ for inputs. A *boolean expression* is a set of literals joined by boolean operators, such as negation $\overline{A}$, conjunction $A \wedge B$, disjunction $A \vee B$, implication $A \rightarrow B$, equivalence $A \longleftrightarrow B$, xor $A \oplus B$, etc.

An expression in disjunctive normal form (DNF) or sum-of-products form is one that is represented as a disjunction of conjunctions of literals, such as $(A \wedge B \wedge C) \vee (A \wedge \overline{B}) \vee (A \wedge \overline{D})$. In the context of machine learning, conjunctions are often regarded as *rules*, and disjunctions of conjunctions, as *rule sets*.

An *intermediate concept* is an expression which models only certain aspects of the target concept. A *hidden feature* is an intermediate concept modeled by a neural network in its hidden layers, which may reduce the dimensionality of the inputs in a way that is significant for the prediction of the outputs.

A *split value* is determined by a neuron $h_{l,n}$ and a threshold value. That is to say that if the same threshold is considered for two different neurons, these are two distinct split values.

The symbology which is used across this work is outlined in Table 1.1.

Symbol	Meaning
$(\mathbf{x}^i, y^i) \in D$	Observation i of some dataset D, made out of feature vector $\mathbf{x}^i$ and class value y^i
$\hat{y}^i$	Class prediction for the feature vector $\mathbf{x}^i$
C_j	Class j for a finite set of classes in the context of a classification problem.
h_l	Hidden layer l if $l >= 1$, input layer if $l = 0$
$h_{l,n}$	The n^{th} neuron of layer l
W^l	Weight matrix between layers $l-1$ and l
$w^l_{j,k}$	Connection between neurons $h_{l-1,k}$ and $h_{l,j}$
$b_{l,n}$	Bias for the neuron $h_{l,n}$
$z^i_{l,n}$	Weighted sum of the activations of layer h_{l-1}
$a^i_{l,n}$	Activation value of neuron $h_{l,n}$ for observation $(\mathbf{x}^i, y^i)$
$\mathbf{a}^i_l$	Activation values of observation x^i for all neurons of layer h_l
$\mathbf{a}^D$	Activation values for neurons of all layers, for all observations in the dataset D
$t^i_{l,n}$	A threshold to divide the range of neuron $h_{l,n}$
$\mathbf{t}_{l,n}$	Set of thresholds to divide the range of neuron $h_{l,n}$
T_l	Set of thresholds to divide each neuron of layer l
$H^j_{l,n}$	Mean of the values that make up cluster j for neuron $h_{l,n}$
$DNF_{h_l \rightarrow c}$	Expression that describes a c with conditions on the activation values of layer h_l
$DNF_{h_0 \rightarrow target}$	Expression that describes the target condition with regards to the inputs

Table 1.1.: Table of symbols.

2 Fundamentals

Classification is a branch of machine learning which poses the task of finding the right class for an observation out of a predefined set of classes. An *observation* is represented by a vector of *feature* or *attribute* values $\mathbf{x}^i = [x_1^i, x_2^i, ..., x_n^i]^T$ and its corresponding class value y^i. A *model* is built which maps feature vectors to class predictions $f(\mathbf{x}) = \hat{y}$, and the goal is that $\hat{y} = y$ for a greater number of unseen feature vectors or so that a cost function is minimised.

A set of examples which have been labeled with the correct class form the *training data* which is used to build the model. The model should be able to capture patterns in this data so as to predict the correct class for unseen feature vectors, yet not learn idiosyncrasies from the training examples and thus have difficulty generalizing to new observations, in which case it is said to *overfit*. As a general rule, the more features make up the observations and the more parameters are used to define the model, the more data is needed to prevent such a shortcoming.

There is a wide array of methods that tackle the classification task. In this chapter, a brief overview is given of *decision trees* and *artificial neural networks*, as these topics are relevant for the incoming chapters.

2.1 Decision Trees

Decision trees are made out of *test nodes* which iteratively partition the feature space and *leafs* which are annotated with class values. A classification is made by starting at the *root* of the tree and transversing it according to whether the characteristic defined in each test is present in the feature vector. The process is deterministic in that there is only one path each feature vector can take.

A popular algorithm used to build decision trees is C4.5 (Quinlan, 1993), which works in the following manner. Starting from the root, a test is selected to divide the training data into disjoint subsets such that more examples in each subset belong to the majority class. Tests usually have the form $x_i \leq threshold$ for continuous attributes and $x_i = value$ for discrete ones. The test t is selected which maximises the normalised information gain (2.2) by dividing the current set into one subset for each outcome of the test. The information gain is the difference in entropy (2.1) between the original and the partitioned sets, which measures the class unevenness within a set. Thus, maximizing the information gain *purifies* the sets by encouraging all members to be of the same class.

$$H = -\sum_{C_j} P(C_j) \log_2 P(C_j) \tag{2.1}$$

$$IG(D,t) = H(D) - H(D|t) = H(D) - \sum_{v \in outcomes(t)} \frac{|D_v|}{|D|} . H(D_v) \tag{2.2}$$

$$D_v = \{(\mathbf{x}, y) \in D | v\} \tag{2.3}$$

Different variants of the algorithm use different criteria to define when to stop branching, but this often occurs when some majority of the examples in the node belong to the same class or no test leads to a positive information gain.

A major advantage of decision trees is that they provide a clear visualisation of the way they work and the path which led to a decision can be easily traced. However, they also have some drawbacks, such as that the amount of data per node decreases as the tree grows, and that they cannot easily model relationships between features. In general, they are highly dependent on the way the attributes are defined, so when the data is highly dimensional some feature preprocessing is usually performed prior to building them.

Decision trees can easily be represented as sets of decision rules, which are also highly interpretable. In contrast with other rule models, the deterministic nature of the tree is maintained and it is not need of additional criteria to clarify ambiguities.

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:

$$X_1 \wedge X_2 \wedge \ ... \ \wedge X_n \rightarrow class$$

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.

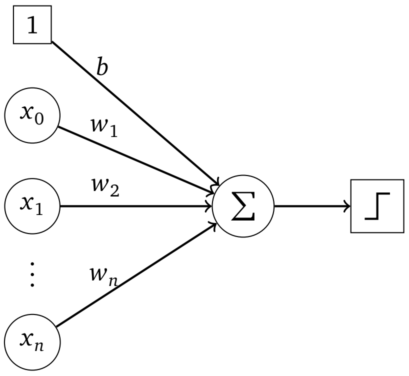


Figure 2.1.: A perceptron neuron.

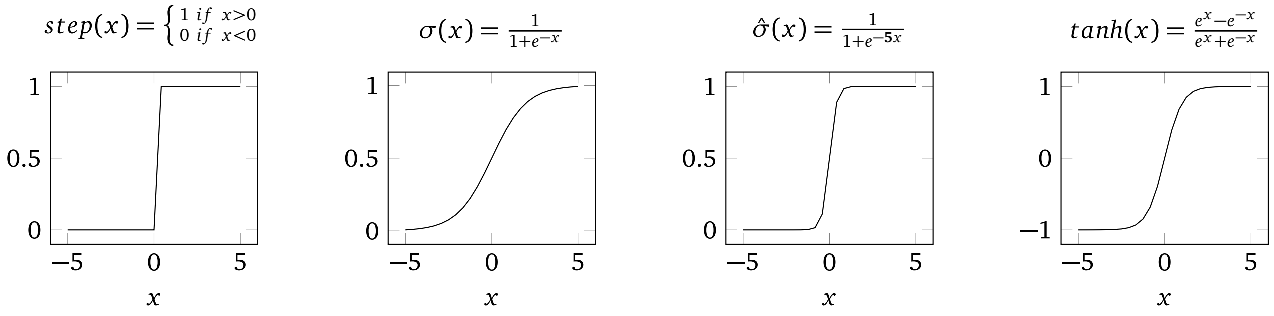


Figure 2.2.: The sign, sigmoid, sigmoid with an increased slope and hyperbolic tangent functions

2.2 Artificial Neural Networks

A theory within neuroscience is that one way in which humans learn and store knowledge is through changes in the dentrite connectivety of the brain. A neuron *fires* if the added excitement level of its incoming connections exceeds a threshold. As a certain neuronal path is activated with more frequency, the strenght of the connections increases so that if neurons at the start achieve an active state, this is likelier to cause those further along the path to be excited (Jesan and Lauro, 2003).

Artificial neural networks emulate this behaviour to some degree, although feedforward structures only make up certain areas of the human brain. They take many forms, but are generally made out of layers of individual units which perform linear transformations of their incoming connections and apply a nonlinearity on top. The backpropagation algorithm presents an effective way to train these models such that nonlinear patterns can be learned within the structures.

2.2.1 The Perceptron

A *perceptron* neuron is a linear classifier which converts an array of binary input values into a binary output. The output is one if the sum of the weighted inputs exceed some threshold, and zero otherwise. Instead of considering the threshold as a separate term, its negated form referred to as *bias* can be regarded as an additional weight which is always multiplied by one, in which case the output is determined by the sign of the sum (2.4). A perceptron neuron is shown in Figure 2.1.

$$\hat{y} = sgn\left(b + \sum_i x_i w_i\right) \tag{2.4}$$

Perceptrons can be stacked by using the outputs of some neurons as inputs of others, thus building relationships between features and modeling more complex functions which are not only linear transformations of the inputs. This forms the base for feedforward neural networks, which is why they are often called *multi-layer perceptrons*.

As the *sgn* function is not differentiable, the parameters cannot be optimised using iterative approaximation techniques. Therefore, instead of applying this function over the weighted sum, other functions are used which create a similar transformation but that are smooth and differentiable. Common choices are the *sigmoid*, which outputs values in the $[0,1]$ range and *hyperbolic tangent*, which extends this range to $[-1,1]$. These functions are illustrated in Figure 2.2.

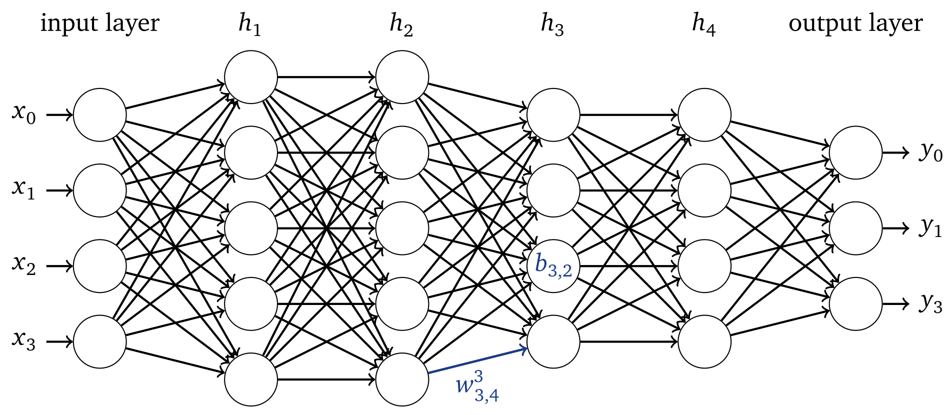


Figure 2.3.: A fully-connected, feedforward, deep neural network with four hidden layers.

2.2.2 Feedforward Artificial Neural Networks

As previously mentioned, artificial neural networks are made out of several layers of stacked neurons. They have an *input* and an *output* layer, as well as at least one *hidden* layer in between. If there is more than one hidden layer, the network is defined as *deep*. Layers closer to the inputs are said to be *shallower*, and those closer to the outputs are *deeper*.

The *activations* $a^i_{l,n}$ of the neurons are calculated starting from the first hidden layer. These are computed by applying the activation function over the weighted sum of the feature values (2.5).

$$a^i_{1,n} = f\left(\sum_k w^{\ 1}_{nk} x_k + b_{1,n}\right) \tag{2.5}$$

The following activations are determined by calculating the weighted sum of the activations from the preceding layer (2.6) and applying the activation function (2.7).

$$z^i_{l,n} = \sum_k w^{\ l}_{nk} a_{l-1,k} + b_{l,n} \tag{2.6}$$

$$a^i_{l,n} = f\left(z^i_{l,n}\right) \tag{2.7}$$

A network is said to be *fully-connected* if all neurons in layer h_i are connected to all neurons of layer h_{i+1}. The *feedforward* characterisation refers to the fact that all connections go in the direction from the inputs to the outputs, which differentiates them from *recurrent neural networks* that do not adhere to this constraint. Within this work, only feedforward networks will be considered.

A fully-connected, feedforward, deep neural network is shown in Figure 2.3. Each arrow between two neurons represents the entry in the corresponding weight matrix that determines the value by which the activation of the shallower neuron are multiplied in the weighted sum. The bias weight is depicted inside the neuron.

2.2.3 Hidden Features

Linear models are effective when applied to linearly separable data, and they have the advantage that finding the best parameter setting is a convex optimisation problem. However, they cannot model data which follows nonlinear relationships unless the features themselves have been designed to capture these patterns.

In contrast, neural networks do not require such preparation, as they can find adequate nonlinear transformations during the training process. In fact, automatically finding relevant *hidden features* in the intermediate layers is one of the main advantages of deep neural networks, and what makes them so well suited to deal with highly dimensional, unprocessed data.

Neural networks can also be leveraged for feature discovery beyond specific classification problems, for instance through the use of auto-encoders. These are networks that approximate the identity function between an input layer and an equivalent (or slightly distorted) output, but which have narrower hidden layers, therefore forcing the network to perform some dimensionality reduction.

Finding such patters is particularly interesting in cases where the input features by themselves do not provide much information. This is for instance the case for image data, where visualisations of the features learned in hidden layers give clear insight into the aspects which are considered when making predictions.

2.2.4 Neural Networks for Classification

For classification problems, there is usually one output neuron per class of interest, and a *softmax* or *normalised exponential* function (2.8) is applied to the last layer in order to convert the output values into probabilities. Each value of an output neuron j thus lies between zero and one and the sum of all output values is always one.

$$P(y^i = C_n|\mathbf{x}^i) = \frac{\exp(z^i_{C_n})}{\sum_{C_j} \exp(z^i_{C_j})} \tag{2.8}$$

A common loss function for classification problems, considering the real class y^i is in one-hot encoding and $\hat{y}^i$ are the probabilities for each class, is the cross-entropy (2.9), which penalises the divergence of the probability estimates from the real class.

$$J(D) = \sum_i \sum_{C_j} -y^i_{C_j} \log(\hat{y}^i_{C_j}) \tag{2.9}$$

2.2.5 The Backpropagation Algorithm

Introducing nonlinearity into the model makes the problem of optimizing the parameters non-convex, so the optimal parameter values cannot be found analytically and, instead, iterative approaches must be used to find an adequate setting.

Probabilistic machine learning models are often trained with variants of *gradient descent*. The basic process consists of calculating the loss $J(\theta^n, D)$ when using the current parameter setting on the training set, or a part thereof, and updating the parameters with the negative gradient of the loss function multiplied by a *learning rate* η (2.10). Naturally, calculating the gradient requires the loss function to be differentiable.

$$\theta^{n+1} = \theta^n - \eta \nabla J(\theta^n, D) \tag{2.10}$$

The backpropagation algorithm, popularised in 1986 by the publication Rumelhart et al. (1988), first makes a *forward propagation* of some train data through the network in order to calculate the overall loss $J(\theta^n, D)$. The error for the output neurons (2.11) is calculated as the Hadamard (element-wise) product $\odot$ between the gradient of the loss function at the output activations and the derivative of the activation function over the linear transformation on each output neuron.

$$\delta_l = \nabla_{\mathbf{a}_l} J(\theta^n, D) \odot f'(\mathbf{z}_l) \tag{2.11}$$

For shallower layers, the error δ_l (2.12) is calculated by considering the contribution of each neuron to the error in the next deeper layer δ_l, where the magnitude of the contribution is determined by the weight connections outgoing that neuron.

$$\delta_l = W^{l+1^T} \delta_{l+1} \odot f'(\mathbf{z}_l) \tag{2.12}$$

It is according to this error that the correction for each paramater is calculated. This process is associated with several challenges, such as keeping the gradient from *vanishing* when training deeper networks, as the distance increases between shallower parameters and the cost. However, these issues are not further discussed in this work.

3 Related Work

Altough in recent years deep neural networks have considerably grown in popularity and there is increased interest in them infiltrating certain domains, artificial neural models have been relevant for several decades, and their lack of interpretability has always been recognised as one of their weaknesses. Therefore, an array of different methods to make them more comprehensible have been developed, some of which are outlined in Section 3.3.

Additionally, this chapter provides an overview of the research in the areas of network pruning (Section 3.1) and discretisation of continuous attributes (Section 3.2), as existing approaches inspire some methods presented in the incoming chapter.

3.1 Training Simpler Models

Training deep neural networks requires a lot of data to prevent overfitting the parameters. This is partly due to the fact that finding the right topology for a network is not a trivial task and iteratively increasing the number of layers and nodes is a time-costly process, so often too-large topologies are chosen.

To avoid this, many techniques have been developed to act as regularizers. A common practice is to include a weighted $L1$ or $L2$ norm of the weights as a penalty term in the cost function to keep these from having big magnitudes. Also, *dropout* (Srivastava et al., 2014) is often used, which introduces noise by temporarily removing connections for an epoch with some probability. There are also many approaches which encourage sparse hidden activations (Ng, 2011) or weights.

In this section, techniques for pruning network connections are reviewed, as this has shown encouraging results in assisting the extraction of rule representations.

3.1.1 Pruning Connections of Neural Networks

Pruning connections or whole neurons of trained neural networks is a common way to adapt the topology of the network to the effective sise of the problem after training, thus discouraging overfitting and increasing the generalisation capabilities. It can also be leveraged to require less time and resources when making classifications (Hassibi et al., 1993; LeCun et al., 1989; Thodberg, 1991).

A connection $w^l_{j,k}$ between two neurons $h_{l-1,k}$ and $h_{l,j}$ can be pruned by equaling the weight entry to zero. This differs from applying dropout in that connections are removed permanently, and which connections to prune is decided according to some criterion, whereas dropout temporarily removes randomly chosen connections for one epoch at a time. Pruning connections also results in pruning whole inputs or hidden units, as a neuron without output connections is disconnected from the network. An illustration of the effect of pruning connections is shown in Figure 3.1.

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 regularisation term in order to minimise the magnitude of the irrelevant weights.

2. One or more weights are pruned, chosen by some heuristic as being the least significant.

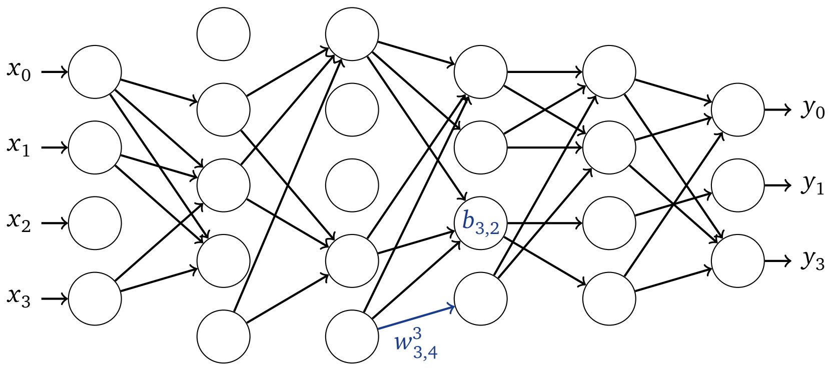


Figure 3.1.: A deep neural network with 61% less connections than the fully-connected equivalent in Figure 2.3.

3. The network is retrained until an acceptable accuracy has been reached or a certain number of epochs have been performed. If the accuracy is acceptable, pruning is resumed; otherwise the last acceptable solution is chosen.

In Thimm and Fiesler (1997), five different heuristics to select the next weight to be pruned are compared. Weight decay is not used for concerns that it may draw the training to a local minimum. The results show that pruned networks often, though not always, achieve a better generalisation. There is, however, no clear negative correlation between the number of maintained connections and the generalisation capabilities. It is also shown that all five pruning heuristics perform similarly well, and the simplest one, which consists of selecting the weights with lowest magnitude, often performs within the best.

In Han et al. (2015), connection pruning is applied to networks with fully connected and convolutional layers trained from image datasets. Weight decay is performed during backpropagation before pruning weights with magnitude below a threshold. After each pruning step, the network is retrained. The results show that pruning iteratively, which is to say performing a retraining phase after each pruning step, allows to discard a considerable higher number of connections. The number of connections could consistently be reduced between nine and thirteen times without loss in accuracy.

3.2 Discretisation Techniques

Many machine learning algorithms which construct symbolic models, including tree and rule set building procedures, train the model by partitioning the data according to adherence of the feature vector to tests. Some approaches are only designed to work on discrete attributes. For those able to handle continuous ranges, usually, *online* discretisation is performed in which a heuristic determines from all possible split values of a continuous attribute x_i the best threshold to partition the data.

The split values considered are often all middle points between any two subsequent values the attribute takes in the train set. When selected in such a manner, the number of times the heuristic is calculated in order to find the best separation grows linearly with the amount of training data, which significantly increases the running time of building the model. Therefore, the effect of instead performing an *offline* discretisation before applying the algorithm, which places an upper bound on the number of split values that are considered, has been extensively studied.

Discretisation methods are usually categorised according to three dimensions (Dougherty et al., 1995). The first distinguishes between *global* approaches which partition all continuous attributes from *local* ones which affect only one. The second differentiates *supervised* methods, which consider class labels for classification problems, from *unsipervised* ones. Lastly, *static* methods divide the range into a set number of intervals whereas *dynamic* ones determine for each attribute the number of intervals it should be divided into.

3.2.1 Unsupervised Methods

The simplest unsupervised discretisation approaches include *equal width binning*, which divides each range into k equally wide intervals; and *equal frequency binning*, where each bin covers the same number of train instances (Catlett, 1991). There are also approaches that use clustering to more closely replicate the distribution of the data (Chmielewski and Grzymala-Busse, 1996).

3.2.2 Supervised Methods

In Berka and Bruha (1998), a supervised offline method for individually partitioning numerical attributes is presented. First, the value each attribute takes in the train set is paired with the classes of the observations which take that value. If the value only occurs for examples of the same class or the frequency of some class is significantly greater, that class is assigned. Otherwise, the value is labeled as *unknown*. Then, the values are ordered and intervals are created that cover the same class. If *unknown* intervals are surrounded by exponents of the same class, then all three are joined. Otherwise, they are grouped with one of the surrounding intervals according to some criterion. The midpoints between interval bounds make up the split values.

When used as a preprocessing step for *C4.5* such that unknown intervals were always joined with the neighbor with the class for which there was a majority, the procedure caused an accuracy decrease of about two points in most cases. Only for one dataset did the accuracy decrease from 90% to 75%. Note that if mixed-class intervals are not joined, the intervals point to the boundary midpoints. In Fayyad (1992), a proof is presented that the split values selected by C4.5 are always in the class boundaries. An example of a situation where mixed intervals are not joined is illustrated in Table 3.1. For this example, the boundary midpoints for the attribute are 1.6, 0.95 and 0.85.

Another supervised method which has shown good results is *RMEP* (Recursive Minimal Entropy Partitioning), presented in Fayyad and Irani (1993). This approach first restricts the initial set of split values to the boundary midpoints so as to assure that the algorithm does not select an unfavorable split value in the first steps. Then, the value that minimises entropy on the current train set is selected, and the search is recursively continued over the partitioned subsets. The procedure stops partitioning when a criterion related to the information entropy of the subsets is reached. The evaluation shows encouraging results, and in Dougherty et al. (1995), this method is found to be the most effective to use as a preprocessing step for C4.5.

x_j	y	$[a, b]$	$C_{[a,b]}$
2	C_0	$[2, 2]$	unknown
2	C_1	$[2, 2]$	unknown
1.2	C_0	$[1, 1.2]$	C_0
1	C_0	$[1, 1.2]$	C_0
0.9	C_1	$[0.9, 0.9]$	C_1
0.8	C_0	$[0.6, 0.8]$	C_0
0.6	C_0	$[0.6, 0.8]$	C_0

Table 3.1.: Example of how the values taken by an attribute can be grouped into intervals in a way that minimises class mixture.

3.3 Extraction of Rule Representations from Neural Networks

Many different approaches have been developed – particularly in the nineties – to extract symbolic representations of neural networks. These techniques greatly differ from each other in several aspects and prioritise different properties of the final representation.

It should be noted that there are also approaches which represent the network using fuzzy rules (Benítez et al., 1997; Huang and Xing, 2002; Kasabov, 1996), although these representations are not considered to be as interpretable as boolean ones (Andrews et al., 1995). Additionally, methods have been developed to detect which attributes and connections are relevant (Olden and Jackson, 2002), as well as to form expressive visualisations, especially for image data, which provide meaningful insight into the way the network operates (Das et al., 2016; Erhan et al., 2009; Zeiler and Fergus, 2014).

3.3.1 Taxonomy of Rule Extraction Approaches

Different categorisations are used to distinguish between methods that extract rule representations from neural networks. One of these refers to the expressive power of the extracted rules, distinguishing between boolean logic and other types of representations, like fuzzy rules. Others regard the computational complexity of the approach, which is particularly relevant for models with more inputs and hidden units, and whether a particular training regime or retraining step is required (Andrews et al., 1995).

Another distinguishing aspect is the transparency with which the network is regarded (Craven and Shavlik, 1994). *Pedagogical* methods treat the network as a black box and map relationships directly between the outputs and the inputs, thus only explaining the predictive behaviour of the model as a whole. In contrast, *decompositional* ones observe the network as a sum of its parts, looking for the contribution of individual parameters or neurons and uncovering hidden features. Approaches which use knowledge of certain elements of the network's architecture but do not explain the parts individually are called *eclectic*.

The methods can also be characterised by the data used to build the model. This is a relevant aspect, as the way the network reacts to implausible feature combinations may not necessarily give insight into how it would classify unseen but naturally occurring observations. Some approaches only use examples the network was trained with. Others use all possible feature vectors, thus building a model which would react similarly to the original network for any hypothetical instance, which is, of cause, only viable when the feature space is small. Lastly, some methods make use of the training set but augment it with generated instances.

Finally, a distinction can be made as to whether the approach works in a *monothetic* or *polythetic* fashion (Fisher and McKusick, 1989). The first group considers each input (be it input feature or incoming connection) separately, while the second regards all attributes simultaneously.

3.3.2 Subset Methods

These methods typically assume a polarisation of the activations and the use of exclusively binary inputs. They search the entire feature space and construct one expression per neuron of interest which describes that neuron as being *active*. Basically, rules are built for a neuron $h_{l,n}$ being in an active state by finding combinations of the incoming weights that cause the bias of the hidden unit to be exceeded. As neurons of layer h_{l-1} are also assumed to be minimally or maximally activate, it suffices to consider the weights instead of the product of the weights with the corresponding activations a_{l-1}.

A shortcoming of such approaches is that observing all subset combinations grows at an exponential rate with the number of incoming connections, which limits their applicability to larger networks because of computational constraints and results in a great number of final rules. To subdue these issues, usually upper bounds are set for the maximal number of literals in each subset.

Another problem is that although the activations of neural networks using most activation functions can be polarised to some degree, it is a hard assumption to consider that any network can be converted to one with only maximally or minimally active neurons while maintaining the initial accuracy. A much more difficult requirement to fulfill is that inputs should be binary, as for continuous attributes to be binarised they first need to be discretised, which usually causes some information loss.

KT

The *KT* algorithm (Fu, 1991) extracts rules from neural networks in a decompositional manner. A neuron is found to be active if its value is greater than some β and inactive if it is smaller than an α. A version of the approach is presented where the slope of the sigmoid is significantly increased during training, which results in an increase in accuracy for one of the two experiments performed.

For each neuron, inputs are divided according to the sign of the connecting weight. As the sigmoid is monotonously increasing and its values lie in $[0, 1]$ it suffices to observe the sign of the weights and the previous activations can be disregarded. The approach then builds the conjunctive rules by finding combinations of the input literals that guarantee a unit being active.

The space of input value combinations is transvered in a breadth-first manner. From the more general rule, which includes the least literals, it is observed whether it accurately classifies all examples it covers. Only if it does not, the rules at the next specificity level are considered. The computational complexity grows exponentially with the number of input features and is controlled by setting an upper layer on the number of literals for each rule.

Polynomial Subset Method

In Tsukimoto (2000) a similar decompositional method is presented, but which keeps the computational complexity of exploring the search space polynomial. Observations with feature vectors of length n are expressed as a multilinear function of n variables (3.1).

$$\sum_{i=1}^{2^n}\left(w_i\prod_{j=1}^{n}e(x_j)\right), e(x_j)\in\{x_j,\overline{x_j}\} \tag{3.1}$$

For each hidden unit, conjunctions of literals are found such that the activation exceeds 0.5. The search starts by regarding only one variable, and allows defining more until the rule is consistent with the neuron being active. The conjunctions are then joined into a DNF, which is later pruned with the train data.

There is no mention of requiring the activations to be polarised, but here as well the rules refer to neurons being active, a state that is defined as exceeding the 0.5 mark (for a sigmoid activation function). The method is said to be applicable to continuous attributes in the $[0, 1]$ interval, but the final rules only present conditions that can be read as a value being *small* or *large*, so an implied discretisation takes place. It is also explicitly stated that a more detailed description of the network's behaviour would require different thresholds to be considered for each unit.

MofN

M-of-N rules are representations where the left side has the form $IF(M\ of\ N)$. If $M = 1$, the rule can be regarded as a disjunction of the literals in N, whereas for $M = N$, the expression is a conjunction. Using this syntax can result in more compact representations. In Towell and Shavlik (1993) the algorithm *MofN* is presented for constructing networks out of previously defined logical constructs, and then extracting M-of-N representations with a subset algorithm.

First, the topology and initial parameters of the networks are determined using *KBANN*, which expresses domain knowledge in symbolic rules, more specifically Horn clauses. A hierarchical structure is constructed to represent the predetermined dependencies, which makes up the base topology of the network by considering each literal as a network node. The connections between nodes which are highly dependent are assigned high initial weights.

Representations for each output and hidden neuron being active are build by finding combinations of weights that exceed the bias. The rule extraction process assumes a polarisation of the activations, as well as that the logic defined in the previous step has not been significantly altered during training. The sigmoid activation function is used and the polarisation is implemented by increasing the slope.

The method starts by clustering the weights of the hidden units and replacing each activation by its closest cluster mean to form equivalence classes. Then, groups that are considered irrelevant by having low weight magnitudes and not many members are pruned. Clusters are pruned if they either cannot have any effect on the unit exceeding the bias even when all members are active or heuristically by regarding the relevance of the cluster when classifying train instances. Afterwards, the bias parameters are retrained while keeping the weights constant.

Lastly, rules are formed which determine, for each cluster, the fractional number of integrands which should be active for the bias to be exceeded. A simplification procedure then transforms each rule into a set of rules where the amounts of members are integers.

The method was able to extract comprehensible and accurate representations when both assumptions were met. However, it showed difficulties when the logic modeled by the hidden units diverged too strongly from the original knowledge. This presents an obvious shortcoming, which is the necessity of sufficiently simple and clearly formulated domain knowledge. Is also illustrates a more general problem of M-of-N representations, namely that while often neural networks naturally repeat similar logic in different neurons, regarding these as equivalent requires some degree of approximation which may cause the accuracy to fall.

3.3.3 Polythetic Methods that Sample Limited Amounts of Generated Data

Under this ample categorisation fall an array of algorithms that attempt to explain all sections of the network in a more flexible manner that subset methods, by sampling the network with a subset of all possible input vectors, chosen according to some criteria.

STARE

STARE (Statistics based Rule Extraction), outlined in Zhou et al. (2000), presents a way to extend subset methods to dataset that contain continuous attributes by creating a new set of observations from the combinations between all possible values of discrete attributes and values of continuous attributes determined by partitioning the original range into a large number of intervals.

The new examples are sampled in the network to obtain the corresponding predictions. Then, attributes are sequentially discretised into as many clusters as necessary for no two examples classified as belonging to different classes to be assigned the same clustered values.

For extracting rules, a subset approach is used which sequentially increments the number of literals, with a maximum of three. For selecting which attributes fall into these three permitted literals, STARE prioritises discrete attributes first and then continuous attributes which are clustered into the least number of clusters.

The extracted rules, which are assigned a priority related to the order in which they were created, were often more accurate than those extracted using C4.5. The authors argument that this is due to the rule sets having captivated more general features from the neural networks.

Augmenting the Training Set

The approach presented in Craven and Shavlik (1994) produces a DNF representation for each class by employing two *oracles*. One provides examples and the other answers whether a conjunctive rule predicts the correct class for all the examples it covers. The first oracle first selects instances from the train set, but then augments this set with an approach similar to that taken in STARE (Section 3.3.3).

The rules take the form of a DNF for each class. The DNFs at any state always correctly classify all current examples, and they are grown successively by creating a new rule each time a new example is added which is not covered by the present rules. The new rule initially replicates the example's feature values, and is generalised by dropping conditions for as long as the second oracle does not mark a decrease in accuracy.

The main difference between this approach and other methods is that instead of describing the behaviour of the network for miscellaneous input combinations, it first considers the examples used to train it and only includes others for as long as the accuracy of the rules increases. It thus avoids an exponential computational complexity while leveraging the possibility to sample the network with unseen instances.

Validity Interval Analysis

The *Validity interval Analysis* technique (Thrun, 1995) finds consistent intervals between any two layers of the network. In the *forward* phase, ranges are defined for the shallow units, and the corresponding range for the deeper units is determined from the shallow constraints. In the *backward* phase, intervals are specified for the deeper units and the shallow intervals are refined. An inconsistency is found if an interval is refined to some $[a, b]$ where $a > b$. If no inconsistency is found, the process *converges*.

The left side of the rules is defined by whether the feature vector falls in an hypercube. The right side specifies that the feature vector $\mathbf{x}$ either belongs or does not belong to a class C_j.

$$If\ \mathbf{x} \in [a_i, b_i]^m \rightarrow C_j$$

$$If\ \mathbf{x} \in [a_i, b_i]^m \rightarrow \overline{C_j}$$

Rules which fall in the first case are verified by finding a contradiction to the constraint that the output value for the j^{th} class is smaller than that of each other class when the inputs are within the bounds of the defined hypercube. For the second type or rules, it is only necessary to reach a contradiction for the specified class.

The process of generating rule sets consists on systematically creating and verifying rules. If all attributes are discrete, the rule search space can be regarded as a tree from the most general to the most specific, and the tree can be transversed from the root until a rule is validated. For continuous input ranges, rules can be found by starting with the rule describing one example and growing the interval of each of the example's attributes for as long as the rule remains consistent.

3.3.4 Monothetic Methods

These approaches regard attributes independently, and only consider those which appear relevant for the performance of the network with respect to some objective. All the methods described in this section aim to explain the behaviour of the neural network using a different machine learning methodology, namely building decision trees.

TREPAN

The pedagogical *TREPAN* algorithm presented in Craven and Shavlik (1995) explains the outputs of the network with respect to the inputs by building decision trees directly between these layers. The tree building process makes use of queried instances, generated from the marginal distribution of each attribute, to keep the amount of data from decreasing as the tree branches. Instead of selecting one feature to perform each partition, an *M-of-N* concept is chosen for each split.

In Milaré et al. (2001), different variants of TREPAN are compared with each other and two other symbolic algorithms, CN2 and C4.5. The last two approaches achieve the best accuracy in the test set when the real class labels are provided, but a TREPAN variant performs better, although the difference is not statistically significant, when the task is to mimic the network and its predictions are treated as class data.

Surprisingly, the implementation of TREPAN which performs best is that which does not make use of additional queries, whereas that which does performs worse that CN2 and C4.5. The authors thus warn against the use of generated instances which may not naturally occur, although this contradicts the results obtained by Zhou et al. (2000).

CRED

CRED (Sato and Tsukimoto, 2001) also builds rules out of decision trees, but only uses the examples in the training data, employ the C4.5 algorithm to build the rules and works in a decompositional manner.

First, a target condition is set which determines the concept for which rules are extracted. This may be related to an output neuron laying in some interval or to it having the highest value. That pattern is used to discretise the class values for all examples.

Then, decision trees are build to explain the discretised output pattern with the hidden units as attributes, using the activation values of the train instances. The activation range of each hidden unit is not divided into being maximally or minimally active. Rather, the online discretisation of the C4.5 algorithm determines where to partition each range. This may result in several thresholds being considered per hidden unit, and on some unit not being regarded at all.

The decision trees are converted into sets of decision rules. Redundant and unsatisfiable rules and terms are deleted, and a post-pruning approach is applied (further described in Section 4.1.3). Afterwards, analogous rule sets are built which explain the split values considered for the hidden units with respect to the inputs. Finally, the *total* rules are formed by substituting the hidden split values with rules which explain the target pattern with respect of the inputs.

By regarding only the relevant shallow neuron states, less rules are created than by other approaches, and a much more fine-grained partition of each neuron's activation range can be performed, as it is not necessary to extract rules for each considered split value. Basically, the approach does not describe the general behaviour of the network but rather gives one explanation for it, namely that which is relevant for the data used to train it.

DeepRED

The *DeepRED* algorithm (Zilke et al., 2016) extends *CRED* to deep neural networks. The process extracts rules between any two layers by building decision trees with C4.5 for layer h_l using the activations from layer h_{l-1} as attributes.

The trees are then converted to DNF representations, and a merging step converts the intermediate representations into one of the outputs using the inputs. Redundant and unsatisfiable rules and conditions are deleted, but unlike in CRED no further pruning takes place.

The algorithm was proven capable of extracting rule representations for deep neural networks, although a high percentage of experiments done on larger datasets aborted due to time and memory constraints, usually during the merging process.

The approach outperformed a pedagogical baseline which neglected the hidden logic and applied C4.5 from the network outputs to the inputs in cases where the data modelled patterns that are difficult to represent in a tree-like fashion. Particularly interesting are the results obtained when DeepRED is applied to a network manually constructed to emulate the parity function with eight inputs. The network has a recursive structure which uses several binary parity structures and is minimally connected (each neuron has a maximum of two incoming and two outgoing connections). The method 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.

There is, additionally, a version of the algorithm that performs a feature selection prior to rule extraction by considering the contribution of each input for correctly classifying the training data and removing inputs that do have a great impact. This proves to be very advantageous when the network is used for high dimensional data.

3.3.5 Connection Pruning as a Preprocessing Step

In Setiono (1997b) a method is introduced to prune connections from neural networks with one hidden layer, thus having two weight matrices W^1 and W^2. First, the networks are trained with a weight-decay penalty.

Connections $w^2_{p,j}$ between the hidden and output layers are then pruned if their weight magnitude is less than a threshold η (3.2), and connections $w^1_{j,k}$ are pruned between the input and hidden layer if their product with any of the deeper connections is low in magnitude (3.3).

If no connection fulfills one of those conditions, then the entry $w^1_{j,k}$ for which the minimum of the maximum products is lowest (3.4) is pruned. Afterwards, the network is retrained. If the final error falls below an acceptable level, the pruning step is repeated; otherwise the last acceptable parameters are restored and the process is stopped.

$$\left|w^2_{p,j}\right| \leq \eta \tag{3.2}$$

$$max_p \left|w^2_{p,j}.w^1_{j,k}\right| \leq \eta \tag{3.3}$$

$$min_{w^1_{j,k}}\, max_p \left|w^2_{p,j}.w^1_{j,k}\right| \tag{3.4}$$

This approach is used extensively – particularly by the author – as a preprocessing step before applying rule extraction algorithms.

The pruning phase is usually followed by a discretisation of the hidden unit activations using clustering. The strength of the clustering is first chosen at random and then iteratively improved so that the strongest discretisation is applied that does not lead to a decrease in accuracy. The method consists of selecting some within-cluster distance ϵ and building an initial cluster with the first activation value $a^0_{l,n}$. The remaining observations either join that cluster or start a new one in the event that no cluster mean is sufficiently close (3.5). If the accuracy of the network does not drop, ϵ is increased, otherwise ϵ is reduced, with the goal of finding the greatest ϵ for each hidden unit that does not cause a drop in accuracy.

$$\left| H^{j}_{l,n} - a^{i}_{l,n} \right| > \epsilon \ \forall j \tag{3.5}$$

In Setiono and Liu (1996) and Setiono (1996), after these steps have been performed, rule sets between layers are built greedily using the discretised activations, and an expression of the outputs to the inputs is found by applying substitution and resolving conflicts. Rules created this way had on average more conditions but less rules than rule sets found using a tree building approach, and the predictive accuracy of the network was at large maintained.

In Setiono (2000), a similar approach to the *MofN* algorithm (Section 3.3.2) is used after applying connection pruning. However, instead of polarizing the activations and only considering the weights, the weights are replaced by either 1 or -1 and it is the activations of the hidden units which are clustered. The inputs are also assumed to be binary. M-of-N representations are only extracted if by replacing the weights in such a manner the accuracy of the network remains acceptable.

Topology Adaptation

The same rule-extraction process is applied in Setiono (1997a) except that following the discretisation step it is observed how many inputs a hidden unit has. If this number exceeds some threshold, the hidden unit is 'split' into as many new units as clusters of activation for the original neuron. Each new unit is treated as an output and a hidden layer is inserted in the middle between the inputs and the new output layer. The network is then retrained and the new subnetwork pruned, and the process is repeated until each neuron only has a small number of inputs. The rule sets extracted in this manner proved to have higher accuracy than those extracted using C4.5, and had less rules.

This method is used in Santos et al. (2000) as part of an approach that genetically searches for the network topology best suited for latter rule extraction. Each individual of the population is determined by the number of nodes in the hidden layer. The initial networks are trained and rules are extracted from them. The fitness score of each individual, which defines the probability according to which the individual will be used for reproduction, is determined by the accuracy and comprehensibility of the extracted rule set. The crossover mechanisms include merging weights, as well as including units present only on the largest participant.

In Fujimoto and Nakabayashi (2003), also, subnetworks are created to replace units with more than six inputs to assist latter rule extraction, although this is not followed by pruning of connections.

In Kamruzzaman and Hasan (2010), the algorithm *REANN* is presented to extract rules, also from networks with one hidden layer. If required by the dataset, the inputs are first discretised. Then, the number of hidden units is determined incrementally, starting with one unit and adding more until an acceptable accuracy is reached. Irrelevant connections are then pruned, and hidden activations are discretised. Finally, rules are constructed from the pruned networks and post-pruning is applied for as long as the accuracy is acceptable.

4 Methodology

The task of extracting representations from deep neural networks which not only explain the network's predictive behaviour but also uncover hidden features poses an array of additional challenges.

The goal of unearthing the relations between hidden layers means the approach should work in a decompositional manner. However, subset decompositional approaches pose too strict requirements on the input features and activation values. Deep neural networks are often trained with very high dimensional data, and the attributes used tend to be continuous. Representing these features in a binary manner would result in information loss and increase the amount of features even further, which makes such approaches only viable if the maximal number of literals per conjunction is kept very low. Also, there is no guarantee that the activations of a network can be polarised to such an extent that only considering one state per neuron would not affect the accuracy.

Therefore, a method similar to the DeepRED algorithm (Section 3.3.4) is employed, as this approach has been proven capable of extracting rule representations from deep neural networks. By only using the train instances, the number of observations to regard does not grow exponentially with the amount of inputs, and there is no fear of extracting logic which may never be evoked for naturally occurring feature vectors.

However, the DeepRED method is mainly focused in finding an accurate and comprehensible expression which explains the outputs in terms of the inputs. Although the intermediate expressions can be analyzed, often too many of these are created, as no limit is placed on the number of thresholds which partition each neuron. Therefore, these intermediate rule sets often do not give an easily interpretable picture of the function of the hidden units.

To affront these issues, the approach is extended with a post-pruning approach (Section 4.1.3) which is applied each time an intermediate expression is extracted, so as to find more accurate and compact intermediate expressions than those extracted directly from the decision trees, as well as between substitution steps to avoid the partial errors from propagating to the final expression $DNF_{h_0 \to target}$.

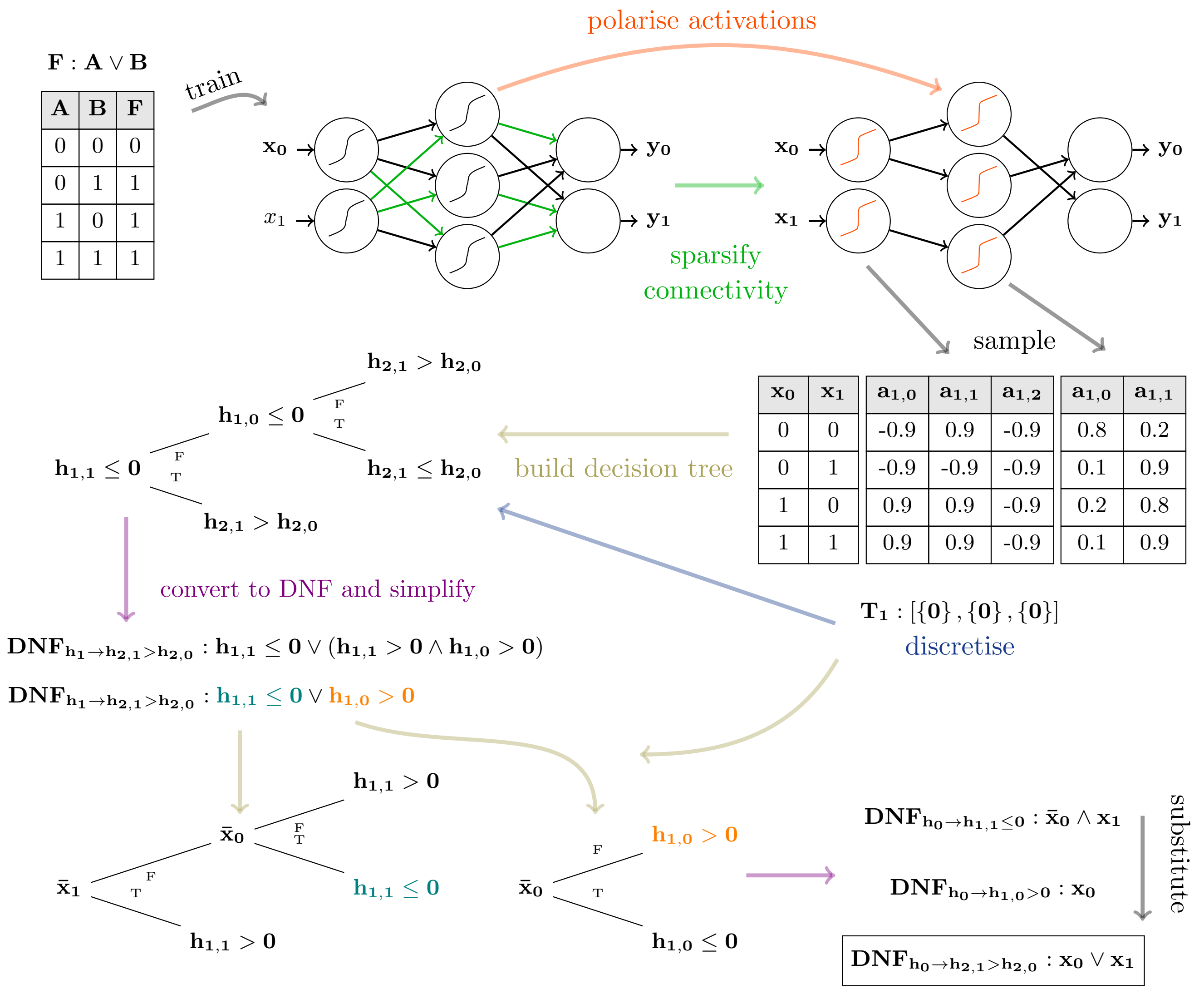
Also, it is explored to what degree applying two different offline discretisation techniques affect the intermediate and final representations. Both methods allow the definition of an upper bound on the number of thresholds per hidden unit. The first, *Activation Clustering* (Section 4.2.1) builds the minimal number of clusters so that the accuracy of the network does not decrease when the activation value of each hidden unit is replaced by the closest cluster mean, and the medium points between cluster boundaries are taken as thresholds. The second, *Layerwise Threshold Preselection* (Section 4.2.2), chooses the thresholds to split all neurons of layer h_l jointly by considering the problem of classifying the train instances as being above or below each relevant threshold for layer h_{l+1}.

The effect is also observed of preceding the rule extraction with two types of network alterations which have been shown to assist the latter extraction of comprehensible representations, namely encouraging sparse connectivity and minimally or maximally active hidden units. These are implemented by the procedures *Weight Sparseness Pruning* (Section 4.3.1) and *Activation Polarisation* (Section 4.3.2), respectively.

The general workflow of reconstructing a boolean expression from a neural network is illustrated in Figure 4.1. First, a dataset is created from all interpretations of the literals, and a network is trained with the data to emulate the predefined concept. The network is then modified so the activations are polarised and irrelevant connections are pruned. Afterwords, the model is sampled to obtain activation values for each training instance. The target concept is defined by class C_1 having a higher probability than C_0. A first tree is built to predict under what activation settings of layer h_1 the target concept is fulfilled. The tree is converted into a DNF representation which is simplified. Then, a tree is built for each literal which appears in the simplified expression, using the input values as attribute data. Each of these expressions is extracted and simplified, and a last step substitutes the literals with regards to layer h_1 from $DNF_{h_1 \to a_{2,1} > a_{1,1}}$ to form $DNF_{h_0 \to a_{2,1} > a_{1,1}}$. Depending on the discretisation procedure used, the threshold values the neurons can be partitioned into are defined before starting the entire rule extraction process (activation clustering) or before building trees for one layer (layerwise threshold preselection).

4.1 Extracting Intermediate and Final DNF Representations

The extraction of the intermediate expressions is illustrated in Algorithm 4.1. It starts by building a model for the target elected to define the binary class value of the examples. This could be done in different ways, such as one or several outputs exceeding or not determined thresholds. A more adequate problem formulation for classification problems is whether the specified output has the highest activation value amongst all in the output layers, which is the target employed in this work. The *isGreatest* function in line 12 determines a binary classification for each observation with respect to this target. Afterwards, in the iterations which build expressions for shallower layers, the class values are determined by whether or not the relevant deeper neuron surpass the corresponding thresholds.


F : A ∨ B
train
polarise activations
sparsify connectivity
sample
build decision tree
discretise
convert to DNF and simplify
substitute
$T_1 : [\{0\},\{0\},\{0\}]$
$h_{2,1} > h_{2,0}$
$h_{1,0} \leq 0$
$h_{2,1} \leq h_{2,0}$
$h_{1,1} \leq 0$
$h_{2,1} > h_{2,0}$
$DNF_{h_1 \to h_{2,1} > h_{2,0}} : h_{1,1} \leq 0 \vee (h_{1,1} > 0 \wedge h_{1,0} > 0)$
$DNF_{h_1 \to h_{2,1} > h_{2,0}} : h_{1,1} \leq 0 \vee h_{1,0} > 0$
$DNF_{h_0 \to h_{1,1} \leq 0} : \bar{x}_0 \wedge x_1$
$DNF_{h_0 \to h_{1,0} > 0} : x_0$
$\boxed{DNF_{h_0 \to h_{2,1} > h_{2,0}} : x_0 \vee x_1}$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

x_0	x_1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,0}$	$a_{1,1}$
0	0	-0.9	0.9	-0.9	0.8	0.2
0	1	-0.9	-0.9	-0.9	0.1	0.9
1	0	0.9	0.9	-0.9	0.2	0.8
1	1	0.9	0.9	-0.9	0.1	0.9

Figure 4.1.: General workflow for reconstructing a boolean formula.

The general process for each layer h_l consists on first determining the split values of the shallower layer h_{l-1} and consecutively extracting expressions $DNF_{h_{l-1} \to h_l > t}$ or $DNF_{h_{l-1} \to h_l <= t}$ for the relevant conditions of layer h_l. Then, the same is repeated for layer h_{l-1} by considering as relevant conditions all those used in deeper expressions.

The main difference between the implementation and how DeepRED proceeds is that the split values for a layer are chosen jointly before the trees are constructed so upper limits can be set on the amount of thresholds per neuron. These are determined either by the *Activation Clustering* (line 8) or the *Layerwise Threshold Preselection* (line 9) procedures. The approaches could, of course, be combined, which is the situation illustrated in the pseudo code.

Once C4.5 has built a tree for each relevant condition using the predefined split values, the tree is represented as a DNF (line 15) which is simplified and pruned (line 16). In the event that the complementary condition has also been used in a deeper expression, the process is repeated (line 20). Note that it is only for the conditions which remain after these changes that rules will be extracted in the following iteration.

The expression $DNF_{h_0 \to target}$ of the target condition with respect to the input values is constructed from the intermediate expression associated with the target condition $DNF_{h_l \to target}$ by consequently substituting the conditions of each conjunction with the corresponding expressions from the intermediate expressions set, and repeating this process until the literals are conditions of the inputs. After each replacement step, the expression is simplified and pruned to minimise the length and reduce error propagation. This process is shown in pseudo code in Algorithm 4.2.

Algorithm 4.1: Build Intermediate Expressions

```
Inputs  : $\mathbf{a}^D$ , target
          clusterMeans                          // If Activation Clustering is performed
Outputs: intermediateExpressions    // Map linking conditions at all network depths to DNFs

1  $T \leftarrow$ activationClustering(D,$\mathbf{a}^D \cup T_0$)
2  intermediateExpressions $\leftarrow$ {}
3  deepConditions $\leftarrow \{target\}$
4  L $\leftarrow$ layer(*target*)
5  while $L > 0$ do
6      shallowActivations $\leftarrow \mathbf{a}^D_{L-1}$
7      shallowConditions $\leftarrow$ {}
8      $T_{L-1} \leftarrow clusterMeans_{L-1}$
9      $T_{L-1} \leftarrow$ layerThresholdPreSelection(*shallowActivations*, $T_{L-1}$, *deepConditions*, $\mathbf{a}^D_L$)
10     foreach $(L, n, t, s) \in$ *deepConditions* do
11         if $L =$ layer(*targetCondition*) then
12             deepClasses $\leftarrow$ isGreatest($\mathbf{a}^D_L$,$n$)
13         else
14             deepClasses $\leftarrow$ classValues($a^D_{L,n}$,t)
15         $DNF_s$,$DNF_{\bar{s}} \leftarrow$ extractRuleSets(C4.5(*deepClassifications, shallowActivations*, $T_{L-1}$))
16         $DNF_s \leftarrow$ postPrune(simplify($DNF_s$),(*shallowActivations, deepClasses*))
17         intermediateExpressions $\leftarrow$ intermediateExpressions $\cup \{((L,n,t,s), DNF_s)\}$
18         shallowConditions $\leftarrow$ shallowConditions $\cup \{c\ \forall c \in conjunction \forall conjunction \in DNF_s\}$
19         if $(L, n, t, \bar{s}) \in$ *deepConditions* then
                       // If the opposite condition is also used in a deeper expression
20             repeat steps 16 to 18
21             deepConditions $\leftarrow$ deepConditions $\setminus \{(L, n, t, s)\}$
22
23     L $\leftarrow$ L-1
24     deepConditions $\leftarrow$ shallowConditions
25 return *intermediateExpressions*
```

4.1.1 Extraction of DNF Formulas from Trees

Each tree determines whether the value of an example for a certain input or hidden unit exceeds a threshold. 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.

In another context, these could be regarded as *if-then* rules which would *fire* if all conditions at the right side were satisfied, in which case the class at the left side would be predicted.

Algorithm 4.2: Get Expression with Input Conditions

```
Inputs  : intermediateExpressions, $\mathbf{a}_l^D\ \forall l \in layers$ , target
Outputs: $DNF_{h_0 \to target}$
1  $DNF_{h_L \to target}$ ← getValue(intermediateExpressions, target)
2  deepClasses ← isGreatest($\mathbf{a}_L^D$,neuron(target))
3  while $L>0$ do
4      shallowActivations ← $\mathbf{a}_{L-1}^D$
5      $DNF_{h_{L-1} \to target}$ ← {}
6      foreach conjunction ∈ $DNF_{h_L \to target}$ do
7          foreach literal ∈ conjunction do
8              conjunction ← substitute(literal, getValue(intermediateExpressions, literal))
9          $DNF_{h_{L-1} \to target}$ ← $DNF_{h_{L-1} \to target}$ ∪ distribute(conjunction)
10     $DNF_{h_L \to target}$ ← postPrune(simplify($DNF_{h_{L-1} \to target}$),(shallowActivations, deepClasses))
11 return $DNF_{h_L \to target}$
```

Within this work, a separate DNF formula is maintained for each literal, so there are two separate DNFs per split value, each of which fires as soon as one of its rules fires. A DNF formula for the event of neuron $h_{1,2}$ exceeding the threshold 0.5 may look like:

$$DNF_{h_0 \to h_{1,2} > 0.5} : (h_{0,1} > 0.5 \land h_{0,2} \leq 0.3) \lor (h_{0,2} > 0.3 \land h_{0,3} > 0.7) \ \to h_{1,2} > 0.5$$

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 $h_{1,2} \leq 0.5$ acted as terms within rules describing the second hidden layer, then one tree would be built and two different expressions $DNF_{h_0 \to h_{1,2} > 0.5}$ and $DNF_{h_0 \to h_{1,2} <= 0.5}$ would be extracted.

These expressions are complementary after being extracted for the tree, but this may no longer hold once simplification and pruning is applied, which may result in both DNFs firing for one example, or none doing so. Usually, additional criteria are needed to determine which class an example would belong to in the events that several rules for different classes were to fire – such as a priority list – or that no rule would fire – such as a default rule.

However, rules are only extracted for one output target class at a time, which is to say that only one expression is maintained for the selected class, predicting the belonging to that class against the belonging to any of the other classes. Therefore, the inconsistencies within intermediate expressions do not translate to ambiguities in the final class prediction.

Preliminary experiments were performed where only one expression per split value was maintained, and that for the complementary literal was considered as its negation. However, this often caused a slight decrease in accuracy and required the negation to be calculated.

4.1.2 Logical Simplification

The process of building the expressions $DNF_{h_0 \to target}$ by sequentially forming each $DNF_{h_{l-1} \to target}$ from $DNF_{h_l \to target}$ through substitution 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 each time a replacement step is carried out. Both CRED and DeepRED (Section 3.3.4) remove 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:

$$h_{0,2} > 0.3 \land h_{0,2} \leq 0.2 \ \to \ h_{1,2} > 0.5$$

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 example expression, the second and third rules are redundant:

$$h_{0,2} > 0.3 \lor (h_{0,2} > 0.3 \land h_{0,4} \leq 0.2) \lor h_{0,2} > 0.4 \ \to \ h_{1,2} > 0.5$$

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

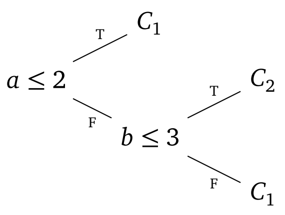


Figure 4.2.: A decision tree for which the DNF representation extracted by joining the paths from the root to the leafs has redundant terms.

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

In addition to the described redundancies, a common pitfall affects DNF representations extracted from decision trees related to the sequential manner in which the tree is evaluated. An example is illustrated in the decision tree in Figure 4.2. The expression extracted from this tree to classify C_1 would be $a \leq 2 \; \lor (a > 2 \;\land\; b > 3) \rightarrow C_1$, 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) and Quinlan (1987b), where an algorithm is presented for converting decision trees into small sets of production rules of the form *IF <conditions> THEN <class> (<certainty factor>)*. The method consists on every condition that does not pass a probabilistic significance test being removed. The resulting rule sets are not equivalent to the original trees but a generalisation of the model. In the experiments performed these rules achieve better accuracy than the trees.

Preliminary experiments were performed that used this type of pruning within the context of extracting rule representations from deep neural networks. However, although the complexity was strongly reduced, the pruning proved to be too aggressive and considerably affected the accuracy of the final and intermediate expressions.

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, the expression

  $$(a > 2 \land b \leq 3 \land c > 2) \lor (a > 2 \land b \leq 3 \land c \leq 6 \land d > 5) \rightarrow C_1$$

  would be transformed to

  $$(a > 2 \land b \leq 3 \land c > 2) \lor (a > 2 \land b \leq 3 \land d > 5) \rightarrow C_1$$

- 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

  $$(a > 2 \land b \leq 3 \land c > 3) \lor (a > 2 \land b \leq 3 \land c \leq 5) \rightarrow C_1$$

  In this case, the complemented condition is deleted and the two rules are merged into one, as

  $$a > 2 \land b \leq 3 \rightarrow C_1$$

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

4.1.3 Rules Post-pruning

After transforming the decision trees into DNF representations, it is possible that very similar rules remain which come from different tree branches. This occurs often when some attributes are dependent on each other, as decision trees cannot capture such patterns in a single test. These rules may not have a redundancy relation but still not provide more information than a simpler rule would.

The danger of ending up with an overly complicated rule set is much greater after performing each replacing step within Algorithm 4.2 while building the final expression. In that context, besides affecting the comprehensibility, this can lead to propagating the error of the intermediate expressions into that of the target condition with respect to the inputs.

However, a too strong simplification should also be avoided, as the repercussions of simplifying intermediate concepts on the final expression cannot be observed until the end.

Merge Between two Rules

The *CRED* algorithm (Section 3.3.4) 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:

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

A rule r for class C_j is found to be 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 (4.1).

$$J(r, C_j) = p(r)p(C_j|r)log_2\frac{p(C_j|r)}{p(C_j)} \tag{4.1}$$

Applying this approach was tested in preliminary experiments and it was found that, although it is successful in reducing the complexity, when used to extract rules from deep neural networks it has a strong negative effect on the accuracy 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 generalising effect.

However, if only merges are performed which increase the accuracy, the benefits of reducing the number of rules and terms can be leveraged without considerably affecting the accuracy. Merging two rules is thus one of the alternative transformations which can be carried out.

Overview of the Post-pruning Approach

A method for reduced error pruning was implemented that only makes a change on the rule set if this positively affects the accuracy towards the specified condition. That is to say, if the expression

$$DNF_{h_1 \rightarrow h_{2,2} > 0.5} : h_{1,1} > 0 \vee (h_{1,2} \leq 0.3 \wedge h_{1,3} > 0.7) \ \rightarrow h_{2,2} > 0.5$$

were to be pruned, the activation values of the training instances for $h_{1,1}$, $h_{1,2}$, $h_{1,3}$ and $h_{2,2}$ would be observed to calculate the consistency between the expression in the left side and the condition in the right one.

Briefly, the proposed approach works as follows. A more detailed representation is provided by the pseudo code in Algorithm 4.3.

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

- The rule is deleted
- A condition is removed from the rule (calculated for all conditions of the rule)
- The rule is merged with another rule from the set (calculated 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 (4.2) (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 accuracy, are calculated. Calculating the varying accuracy 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.

$$P(r) = \frac{true\ positives}{true\ positives + false\ positives} \tag{4.2}$$

For each rule, the modification which leads to the highest accuracy is chosen, and if it is higher than when leaving the rule unchanged, the change is performed. Unless the modification consists on removing the rule completely, the precision

Algorithm 4.3: Rule Set Post-pruning

```
Input  : DNF, pruning set $X$
Output: pruned DNF
1  remainingRules ← DNF
2  lastAccuracy ← accuracy($X$, $DNF$)
3  unseenRules ← heap({(precision($r$), $r$) | $r \in DNF$ })
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         accuracy$_r$ ←accuracy($X$, remainingRules \ {$rule$} ∪ {$r$})
11     newRule ← $argmax_{r \in \text{ruleVariants}}$ accuracy$_r$
12     maxAccuracy ← $max_{r \in \text{ruleVariants}}$ accuracy$_r$
13     if maxAccuracy > lastAccuracy then
14         lastAccuracy ← maxAccuracy
15         remainingRules ← remainingRules \ {rule}
16         if newRule ≠ emptyRule then
17             unseenRules ← unseenRules ∪ {(precision($newRule$), newRule)}
18             remainingRules ← remainingRules ∪ { newRule}
```

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.

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 resemblance to the network was higher when all the available data was used to build the decision trees, and the same data was used for post-pruning.

4.2 Discretisation

A central aspect when contemplating properties that make rule representations for the intermediate layers of deep neural networks comprehensible is the number of thresholds considered per unit. The behaviour of a neuron which can be described in terms of being below or above one dividing value is naturally much easier to comprehend than if a different logic is described for an array of activation ranges.

If only the online discretisation done by C4.5 is performed, which chooses the thresholds that best split the train data for each decision tree separately, it is not possible to put an upper bound on the number of thresholds per neuron and select the split values which adhere to that constraint and best partition the neurons of a layer. Also, online discretisation requires all possible split values to be considered for each recursion step during tree building.

As several algorithms for extracting rule representations from neural networks employ some form of discretisation (Section 3.3), and this appears to greatly help the extraction task, in the context of this work two different discretisation techniques are explored. The first consists of clustering the activation ranges as long as the accuracy of the network is not affected. The second attempts to find the thresholds for neurons of layer h_l that best separate the data as being below or above each split value selected for layer h_{l+1} before building the decision trees between layers h_{l+1} and h_l.

4.2.1 Activation Clustering

In the approaches outlined in Section 3.3.5 that leverage connection pruning for rule extraction, the pruning phase is followed by discretisation of the activation values. In Setiono (1996), a proof is presented that as long as the within-cluster distance is sufficiently small, the activation values of each unit can be discretised without loss in accuracy. Also, it is stated that the complexity of the final rules is proportional to the number of activation clusters.

The effect of such a transformation is evaluated as part of the rule extraction process. However, as the original method is only applicable to neural networks with one hidden layer, it is altered to work on deeper networks. Also, Lloyd's iterative *k-means* algorithm (Lloyd, 1982) is applied for the clustering, and instead of increasing within-cluster distance, the number of clusters k is decreased.

The method starts by selecting an initial number of clusters k and clustering the activations of each neuron of the network into k groups. Then, starting from the last hidden layer and considering all units of that layer separately, the number of cluster means is reduced by one unit and the means are recalculated. Note that this number may be lower than k if there are less distinct activation values that neuron takes.

If this causes a drop in accuracy, the previous cluster means are restored, otherwise k is reduced again. Once a satisfactory discretisation is found for all neurons of that layer, the process is repeated with the adjacent shallower hidden layer, and so on until the first hidden layer has been discretised. When a forward pass is made, each activation is replaced with the cluster mean that is closest in terms of euclidean distance (4.3).

$$\hat{a}^i_{l,n} = argmin_{H^j_{l,n}} \left| H^j_{l,n} - a^i_{l,n} \right| \tag{4.3}$$

In Sato and Tsukimoto (2001), it is argued that this discretisation approach is not used within the *CRED* algorithm because the method does not extract rules for conditions that are not of interest for rules of a deeper layer, and some cluster means may not be relevant. Instead, the pruning method outlined in Section 4.1.3 is preferred to indirectly reduce the number of split points by removing conditions from the rules.

However, as *CRED* is applied on networks with only one hidden layer, there is no danger that selecting too many split values will have an exploding effect on the number of intermediate expressions that must be extracted from the shallower layer.

In the context of this work, rule representations are not extracted for each cluster mean. Instead, this it performed as a preprocessing step to select the split values that will be provided to the *C4.5* approach, which are the midpoints between subsequent cluster boundaries. This method thus provides a way to restrict the number of thresholds considered per neuron which acts globally for all trees built between two layers.

4.2.2 Layerwise Threshold Preselection

The second approach is inspired by the *RMEP* algorithm (Section 3.2.2).

First, a dataset is created where the attributes are the activation values of layer h_i, and the classes are a concatenation of boolean values, with one character assigned for each split value that has been deemed relevant for layer h_{i+1}. An example is illustrated in Table 4.1 of how the entry for an the observation x^o for the preselection of split values of layer 3 could look like, made out of the shallow activations and the placing of the deep activations with respect to the split values previously determined for the deep layer.

After defining the dataset in this manner, a simulated tree building procedure is applied which separates the data, described in pseudo code in Algorithm 4.4. The starting split values that can be selected are the class boundary midpoints, which is to say the midpoint between two consecutive attribute values of different classes.

This is similar to the regular C4.5 in that the split value that leads to the greatest normalised information gain is selected (line 8), and in that the tree grows recursively (line 18).

However, the thresholds are stored instead of a tree structure, and they are accessed globally so that, if two split values lead to the same gain, that one is chosen which has already been selected in another branch (line 10) or that belongs to a neuron (line 11). Also, the growth of the tree is not restricted by any parameter other than no gain being found.

$a^0_{3,0}$	$a^0_{3,1}$	$a^0_{3,2}$	$a^0_{3,3}$	$a^0_{4,0}$	$a^0_{4,1}$	$a^0_{4,2}$
0.465	0.876	0.134	0.271	0.826	0.152	0.443

$\mathbf{t}_{4,0}$	$\mathbf{t}_{4,1}$	$\mathbf{t}_{4,2}$
$\{0.511\}$	$\{\}$	$\{0.312, 0.687, 0.575\}$

a^0_0	a^0_1	a^0_2	a^0_3	class
0.465	0.876	0.134	0.271	1011

Table 4.1.: Example of a dataset for performing layerwise threshold preselection

4.3 Retraining of Deep Neural Networks

In the following section, the approaches are presented which are used to retrain a deep neural network in order to encourage sparseness in the weight matrices or make the hidden units be either maximally or minimally active.

Algorithm 4.4: Simulated Tree Builder

```
Global : T               // Thresholds for a layer, where $T_n$ are those for neuron $n$
Inputs : classCutPoints     // Middle of consecutive values of different classes for each
         neuron
         max                                        // Maximum thresholds per neuron
         data                                 // Dataset as exemplified in Table 4.1
1  if $|data| = 1$ then
2      return
3  bestGain, bestAttribute, bestThreshold ← 0, 0, 0
4  bestTrue, bestFalse ← {}, {}
5  foreach $attribute \in |classCutPoints| : |T_{attribute}| < max$ do
6      foreach $threshold \in classCutPoints_{attribute}$ do
7          trueSet, falseSet ← splitOn(data, attribute, threshold)
8          gain ← normalizedInfoGain(trueSet, falseSet)
9          if $gain > bestGain$ |
10         $(gain = bestGain$ & $threshold \in T_{attribute}$ & $bestThreshold \notin T_{bestAttribute})$ |
11         $(gain = bestGain$ & $|T_{attribute}| < |T_{bestAttribute}|)$ then
12             bestAttribute ← attribute
13             bestThreshold ← threshold
14             bestTrue, bestFalse ← trueSet, falseSet
15 if $bestGain = 0$ then
16     return
17 $T_{bestAttribute} \leftarrow T_{bestAttribute} \cup \{bestThreshold\}$
18 simulatedTreeBuilder(classCutPoints, trueSet)          // Grow simulated tree
19 simulatedTreeBuilder(classCutPoints, falseSet)
```

Within this work, the accuracy on the entire dataset is considered to guide the retraining, as the goal is to train the networks to exactly emulate predefined concepts. However, in another setting it would be advisable to not regard the accuracy on the data for which the loss is calculated but on a separate validation set which is not used to optimise the parameters. Although both modifications are likelier to prevent overfitting than to promote it, using a validation set gives a truer assessment of the performance of the network.

4.3.1 Weight Sparseness Pruning

The network pruning technique is similar to the connection pruning approaches described in Section 3.1. In contrast to those methods, it 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 is generally lower, 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 will be 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 parity function with eight inputs in Zilke et al. (2016). 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.

Preliminary experiments show that 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 network. When rules are extracted from such networks using a reduced subset of combinations, intermediate rules are extracted that overfit to the currently available examples, and concentrate 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. Therefore, the accuracy on the unseen instances never exceeds fifty percent and actually decreases as more training data is presented (a phenomenon that also affects C4.5).

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.

General Methodology

The pseudo code for the proposed method is depicted in Algorithm 4.5.

A connection $w^l_{j,k}$ is represented by the index of the weight matrix l and the row and column indexes j and k. The number of entries that have already been pruned in each row or column of each weight matrix is maintained in order to calculate the *neighborhood sparseness* of the remaining entries. This value is determined by, for an entry $w^l_{j,k}$, the sum of entries that have been pruned in row j of matrix W^l plus those that have been pruned in column k 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 6) 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 resorted. Otherwise the last accepted parameters are restored and the next least sparse connection is removed from the list.

Iterations Used for Retraining

Preliminary experiments show that it becomes clear during a small number of iterations if a 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 epochs.

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 initialised 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 23).

Re-exploration of a Connection

It was also 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.

4.3.2 Activation Polarisation

The fact that the activation range of the hidden units is continuous has several negative repercussions, such as making it more costly to classify new instances. Techniques have been developed for binarizing the parameters and activation values (Aizenberg et al., 2013; Courbariaux et al., 2015). However, most networks are trained in such a way that the hidden units can take any value within the range.

Algorithm 4.5: Weight Sparseness Pruning

```
Input  : network, trainData, maxNonImprovements, maxSteps, iterations, accuracyDecrease
Output: pruned network
targetAccuracy ← network.getAccuracy (trainData)*(1-accuracyDecrease)        /* use separate
 validation set if enough data available */
parameters ← network.getBiasWeights
connections, prunedInRow, prunedInColumn ← initConnections (network.arch ())
while connections ≠ {} do
    nonFound ← True
    connections ← sparsitySort (connections, prunedInRow, prunedInColumn)
    forall $w^l_{j,k}$ ∈ connections do
        connections ← connections \ {$w^l_{j,k}$}
        network.pruneConnection ($w^l_{j,k}$)
        lastAccuracy ← network.getAccuracy (trainData)
        nonImprovements ← 0
        runs ← 0
        while nonImprovements < maxNonImprovements & steps < maxSteps & nonFound do
            for k ∈ [0, iterations) do
                network.train (trainData)
            steps ← steps + 1
            accuracy ← network.getAccuracy (trainData)
            if accuracy>=targetAccuracy then
                                   /* the decrease in accuracy is acceptable, hence prune */
                $\text{prunedInRow}_{l,j}$ ← $\text{prunedInRow}_{l,j}$ + 1
                $\text{prunedInColumn}_{l,k}$ ← $\text{prunedInColumn}_{l,k}$ + 1
                parameters ← network.getBiasWeights
                nonFound ← False
            else if accuracy > lastAccuracy then
                lastAccuracy ← max (accuracy, lastAccuracy)
                nonImprovements ← 0
            else
                nonImprovements ← nonImprovements + 1
        if nonFound then
            network.setBiasWeights (parameters)
            network.restoreConnection ($w^l_{j,k}$)
        else
            break                      // break from forall so connections are reordered
return network
```

As representing each neuron with only one expression is a more comprehensible way to illustrate that neuron's purpose in the network that if different expressions have to be regarded for an array of activation intervals, many decompositional rule extraction approaches reduce the possible states each neuron takes to being either at the bottom or the top of the activation range (Section 3.3). In order to extract rules which are true to the network while making this assumption, the networks are trained in a way that the activations are polarised.

There are several ways to achieve this. In the context of this work, a simple retraining step is proposed. Within these retraining epochs, an additional term is added to the loss function in the form of the *KL divergence*. This term is similar to that used in Ng (2011) to encourage sparse activations, but it penalises the divergence of the mean absolute value of each activation (4.4) from a ρ close to one (as the hyperbolic tangent function is used) over all hidden units (4.5).

$$\hat{\rho}_{l,n} = \frac{1}{|D|} \sum_i \left| a^i_{l,n} \right| \tag{4.4}$$

$$\sum_{l,n} KL(\rho || \hat{\rho}_{l,n}) = \sum_{l,n} \rho \log \frac{\rho}{\hat{\rho}_{l,n}} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_{l,n}} \tag{4.5}$$

This term introduces the additional parameters ρ for the optimal activation average and β for the weighting of the penalty term. Instead of having to define β, the retraining method as implemented for this work starts by setting $\beta = 0$. This value is iteratevely increased, performing some number of epochs between each increment, for as long as there is no decrease in accuracy and the divergence stays above some threshold. The last weight and bias parameters are stored before each increase of β, and if the process stops because the accuracy falls, the parameters which were saved last are restored.

5 Experimental Setup

In the following sections, the process which was used to generate the boolean functions that were later reconstructed is described, and these functions are presented. Also, some characteristics are specified of the methodology for training the networks, as well as of the models themselves. An overview is also given of what properties of the extracted representations are to be assessed during the evaluation.

5.1 Topology and Initial Training of the Deep Neural Networks

The topology of the deep neural networks consists of three hidden layers. The first layer has twice as many neurons as there are inputs. The second one has the average of that number and the output width. Lastly, the third hidden layer has the same width as the output. The number of hidden nodes per layer for each dataset can be found in Table 5.3.

The networks use the hyperbolic tangent activation function and a softmax function in the final layer. They are trained – and later altered to promote activation polarisation and sparse connectivity – using the *TensorFlow* framework, with an optimiser that implements the *ADAM* algorithm (Kingma and Ba, 2014) [1] for ten thousend epochs using cross-entropy as the loss function.

The training is done with the entire dataset until a perfect accuracy is achieved. Thus, it is assumed that the network models the original formula, although no further statement can be made regarding in what way the logic is spread out within it.

5.2 Data Base

The boolean functions used to train the networks and which are later reconstructed are generated as follows. The number of maximum literals the expression may have, as well as a *grouping* parameter and the starting operator – either *AND* or *OR* – are determined. The defined number of groups is formed, with each randomly selecting a subset of the available literals.

When selecting the first group, each literal has the same probability of being selected, but before those that make up the next group are chosen, the probability of the used literals is reduced 50%, and that of the remaining literals is appropiately increased. The literals can thus be shared between groups, but this is discouraged during the generation process so the final expression is not simplified to a much smaller one.

It is then randomly decided whether each aggregation will be negated, with a probability of negation of 0.2. The possibly negated groups are joined with the elected starting operator, and the procedure is repeated recursively within each group, until no more subdivisions can be performed. The *AND* and *OR* operators are alternated over group levels.

Figure 5.1 exemplifies how such a function may be chosen with five initial literals, a *grouping* parameter of two, and *AND* as a starting operator. From the initial set of literals, two subsets are built by randomly choosing between one and all available literals. This is repeated recursively until only one literal remains in each group. Then, the literals are joined by the respective operators. The resulting expression would be $((\overline{E \lor \overline{D}} \land (\overline{E} \lor \overline{E})) \lor A) \land (C \lor (\overline{A \land \overline{A}}))$.

The expression is then reduced to its minimal DNF form using an implementation of the *EXPRESSO* algorithm (Brayton et al., 1984).[2] The dataset corresponding to the expression is made out of all combinations of literals which appear in the simplified version. This step is crucial so as not to give the approaches which perform weight sparseness pruning an unfair advantage, as otherwise the connections outgoing irrelevant literals would be identified as such and non of these inputs would be errouniously included in the final expression.

The expression from the example in Figure 5.1 would be reduced to $(D \land \overline{E}) \lor A$, and a dataset of eight total examples consisting of all interpretations for three literals would be constructed.

In the context of this work, expressions are chosen with a maximum of twenty literals and a grouping of three. Formulas are discarded which are made out of less than five literals or where less than ten examples belong to each class in the dataset, so that each fold has at least one representative of each class.

The dataset base is made out of 19 expressions chosen thusly, plus the parity function with eight inputs. These are shown is Table 5.1. The expressions are represented with the disjunction (A ∨ B), conjunction ($A \land B$), exclusive or ($A \oplus B$) and negation ($\bar{A}$) operators. Additional information can be found in Table 5.2, such as the number of observations and attributes (determined by the number of literals) and the class distribution.

[1] This approach does not require to specify the learning rate.

[2] The result is not guaranteed to be a global minimum, but is very close in practice and its runtime does not grow exponentially with the number of literals, which affects other simplifications methods like *Karnaugh maps* and *Quine-McCluskey*.

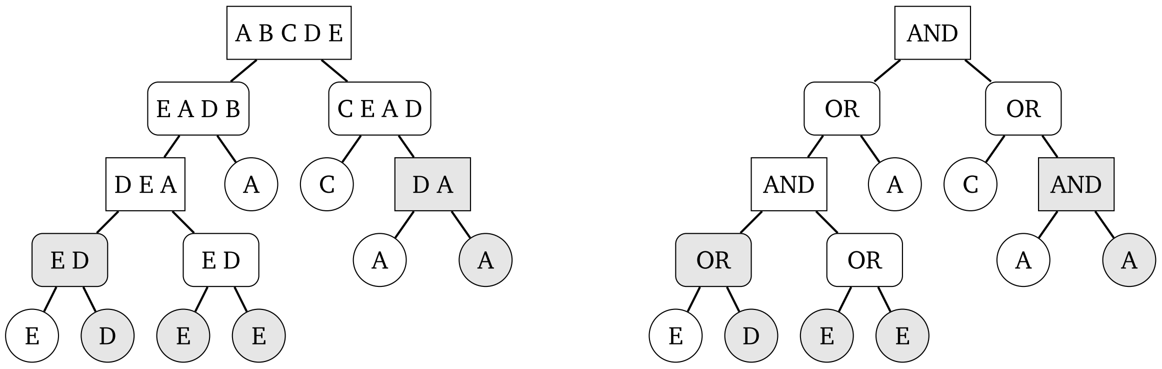


Figure 5.1.: Example of how a boolean function may be generated. The rounding of the corners shows whether the joining operator is *OR* or *AND*, and the shattering marks that the group was negated. The representation in the left illustrates how the literals which form each group are chosen, and that in the right shows the operators which recursively join the subgroups.

Name	Boolean Formula
$F1$	$F \lor G \lor \left(\bar{E} \land \bar{R}\right) \lor \left(A \land \bar{H} \land \bar{J} \land \bar{P} \land \bar{S}\right)$
$F2$	$\bar{E} \lor \bar{F} \lor \bar{Q} \lor \left(G \land \bar{M} \land \bar{R}\right) \lor \left(C \land D \land \bar{J} \land \bar{S}\right)$
$F3$	$(K \land Q \land T) \lor \left(\bar{K} \land \bar{M} \land \bar{P} \land R \land \bar{T}\right)$
$F4$	$(C \land Q) \lor \left(B \land G \land N \land \bar{R} \land \bar{T}\right) \lor (J \land L) \lor \left(B \land N \land \bar{P} \land \bar{T}\right)$
$F5$	$G \lor T \lor \left(\bar{D} \land Q \land S\right) \lor \left(P \land \bar{S}\right) \lor (N \land Q)$
$F6$	$D \lor \left(\bar{E} \land \bar{L} \land N \land T\right) \lor \left(\bar{E} \land G \land \bar{L} \land T\right)$
$F7$	$\left(\bar{A} \land \bar{I} \land \bar{J} \land Q\right) \lor \left(\bar{A} \land \bar{I} \land \bar{J} \land L \land \bar{R} \land T\right) \lor \left(\bar{A} \land \bar{I} \land \bar{J} \land P\right) \lor \left(\bar{A} \land E \land \bar{I} \land \bar{J}\right) \lor \left(\bar{A} \land \bar{I} \land \bar{J} \land L \land S\right) \lor \left(\bar{A} \land B \land \bar{I} \land \bar{J} \land L\right)$
$F8$	$\left(B \land \bar{F} \land J \land \bar{L} \land \bar{M}\right) \lor \left(\bar{N} \land \bar{P}\right) \lor \left(B \land \bar{F} \land J \land \bar{M} \land \bar{R}\right)$
$F9$	$\bar{B} \lor P \lor \left(\bar{C} \land \bar{E} \land J \land \bar{K}\right) \lor \left(\bar{A} \land N \land \bar{R}\right) \lor \left(\bar{H} \land O \land Q \land T\right) \lor \left(\bar{H} \land O \land Q \land S\right)$
$F10$	$\bar{G} \lor H \lor (J \land L) \lor (L \land N) \lor \left(\bar{B} \land \bar{F}\right)$
$F11$	$\left(\bar{N} \land \bar{P}\right) \lor \left(\bar{J} \land \bar{Q}\right)\left(A \land \bar{D}\right) \lor \left(\bar{D} \land \bar{P}\right) \lor \left(A \land \bar{N}\right) \lor \left(\bar{D} \land O\right) \lor \left(\bar{N} \land O\right) \lor \left(\bar{N} \land \bar{Q}\right) \lor \left(A \land \bar{J}\right) \lor \left(\bar{J} \land \bar{P}\right) \lor \left(\bar{J} \land O\right) \lor \left(\bar{D} \land \bar{I}\right) \lor \left(\bar{I} \land \bar{J}\right) \lor \left(\bar{I} \land \bar{N}\right) \lor \left(\bar{D} \land \bar{Q}\right)$
$F12$	$\bar{K} \lor \left(B \land J \land \bar{O}\right) \lor \left(\bar{I} \land \bar{O} \land \bar{T}\right) \lor \left(I \land N \land \bar{O} \land T\right) \lor \left(I \land \bar{O} \land \bar{R}\right)$
$F13$	$\left(\bar{A} \land \bar{B} \land \bar{S} \land \bar{T}\right) \lor \left(\bar{B} \land D \land \bar{T}\right) \lor \left(\bar{B} \land C \land F \land \bar{T}\right)$
$F14$	$\left(G \land J \land K \land \bar{N}\right) \lor (A \land G \land J \land K) \lor (F \land G \land J \land K \land S)$
$F15$	$D \lor \bar{F} \lor \left(H \land P \land \bar{R}\right) \lor \left(H \land \bar{L} \land P\right)$
$F16$	$\left(D \land J \land K \land \bar{L}\right) \lor \left(C \land \bar{F} \land J \land \bar{L}\right) \lor \left(C \land J \land \bar{L} \land O\right) \lor \left(B \land C \land J \land \bar{L}\right) \lor \left(B \land \bar{I} \land J \land \bar{L}\right)$
$F17$	$J \lor \left(\bar{H} \land \bar{L}\right) \lor (N \land O \land P) \lor (K \land N \land O) \lor (D \land N \land O) \lor (E \land H)$
$F18$	$\left(C \land H \land \bar{J} \land S\right) \lor \left(D \land F \land \bar{J} \land S\right) \lor \left(C \land F \land \bar{J} \land S\right) \lor \left(C \land D \land \bar{J} \land S\right) \lor \left(F \land H \land \bar{J} \land S\right) \lor \left(D \land H \land \bar{J} \land S \land T\right)$
$F19$	$\left(B \land C \land \bar{D} \land \bar{H} \land I \land \bar{J} \land L \land \bar{M}\right) \lor \left(B \land C \land \bar{D} \land \bar{H} \land I \land \bar{J} \land \bar{M} \land P\right) \lor \left(\bar{A} \land \bar{R} \land \bar{T}\right)$
$F20$	$A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H$

Table 5.1.: Functions which compose the data base.

5.3 Split of the Datasets

Two different aspects are explored in the experiments. The first considers the situation where the train data directly models the patterns which are to be extracted. That is to say that the entire dataset is used for rule extraction. Here, the

Formula	# attributes	# observations	# obs. C_0	# obs. C_1
$F1$	9	512	419	93
$F2$	10	1024	919	105
$F3$	6	64	10	54
$F4$	10	1024	493	531
$F5$	7	128	112	16
$F6$	6	64	35	29
$F7$	11	2048	237	1811
$F8$	8	256	73	183
$F9$	14	16384	13339	3045
$F10$	7	128	113	15
$F11$	8	256	217	39
$F12$	8	256	171	85
$F13$	7	128	23	195
$F14$	7	128	13	115
$F15$	6	64	51	13
$F16$	9	512	86	426
$F17$	9	512	412	100
$F18$	7	128	21	107
$F19$	12	4096	533	3563
$F20$	8	256	128	128

Table 5.2.: Number of observations and attributes and class distribution for the datasets built from the generated boolean functions

goal is to observe the quality of the intermediate expressions with regards to the interpretability, as well as the maximum accuracy that is achieved if certain constraints are enforced.

The second comparison observes to what extent the approaches are able to extract logical patterns in the hidden layers which are not directly observable in the train data. Here, the accuracy is regarded in a test set made out of the instances not included in the train data.

For this, the data is split into ten, four or two stratified folds and a cross-validation is performed. Also, the situation is considered when there are higher percentages of train data by inverting the respective folds. This results in ten experiments where 10% and 90% of the data is used, four with 25% and 75% being available, and two for the 50% case. For each percentage, the evaluation measure is averaged over all experiments.

5.4 Evaluation Measures

As previously stated, a part of the evaluation is concerned with the discovery of interpretable intermediate concepts that give insight into the inner workings of the network, whereas another is considered for the expressions extracted using a subset of the data.

For the comparison of the quality of the intermediate concepts, the following measures are used:

- The intermediate number of conditions. That is to say, the total number of conditions among all extracted expressions, including those referring to the inputs, regardless to whether the conditions are repeated.
- Number of expressions extracted from the last until the first hidden layer, excluding the expression built to describe the output (as this number is one for all cases). This measure illustrates how many different expressions overall would have to be observed if one wanted to analyse the intermediate concepts.

Also the accuracy (5.1) in the train data is regarded.

Concerning the second part of the evaluation, the accuracy (5.1) in the unseen instances is considered, which compares the classifications of the extracted formula with that of the original one on the test data.

$$accuracy(DNF, D) = 1 - \frac{1}{|D|}\sum_i \left|y^i - \hat{y}^i\right| \tag{5.1}$$

A common measure to evaluate the performance of approaches that find alternative representations for neural networks is the *fidelity* (Towell and Shavlik, 1993; Zhou et al., 2000; Zilke et al., 2016), which compares the predictions of the extracted models with those of the network. The reason this measure is not considered in this work is that the initial networks used here model the target concepts with perfect accuracy, so the degree of similarity to the original network is the same as that to the class values in the dataset.

This, of course, does not hold for the networks after going through connection pruning, which are up to one percent less accurate. However, considering the similarity to the modifined networks would give the approaches that employ this technique an unfair advantage. Although pruning connections has been shown to improve the predictive accuracy of the networks, this evidence is not enough to assume that it would always be more advantageous to emulate the behaviour of the alterned network.

5.4.1 Determining a Significant Difference

For determining when a difference in performance between two classifiers is statistically significant, the *Sign* and *Wilcoxon Signed-Ranks* tests are used, as they are non-parametric and make no assumptions on the distribution of the data, so they are often used to compare models built by two different classifiers on an array of miscellaneous datasets. Both tests are considered for a $p = 0.05$ significance level, and in the event of an uneven amount of ties, the number N of datasets is reduced by one. Also subtracted from this number are comparisons which cannot be performed because of uncompleted experiments.

The first test counts the number of times an approach *wins* or *loses* against the other. It disregards the magnitude of the wins or losses and attempts to disprove the hypothesis that a win or loss is equally likely for each approach. Ties are divided equally between both cases. The critical values considered are those outlined by Demšar and Schuurmans (2006).

The second test does observe the extent to which the performance of an approach differs from the other on each dataset. The differences in the measure of interest are ordered with increasing magnitude, and each dataset is paired with a *rank* value corresponding to that ordering. For each approach, the ranks of the datasets for which it won are summed up and the sums are compared. If several differences are equivalent in magnitude, the rank is averaged (for instance, if those which would fall in places 2 and 3 are the same, 2.5 would be counted for each) and if there are ties, these are distributed equally between the approaches. For performing this test, the implementation in the *Coin* library for the *R* framework is used (Hothorn et al., 2016).

5.5 Approach Variants

For the first part of the evaluation, a total of 30 configurations are compared to assess which modifications help extract the most comprehensible intermediate concepts while not affecting the accuracy of the final expression in the train set.

The first dimension depends on the retraining of the DNN, for which the following options are explored:

- No retraining takes place
- Weight sparseness pruning is performed (Section 4.3.1)
- The activations are first polarised (Section 4.3.2), and then weight sparseness pruning is performed

The second distinction refers to the discretisation, for which there are the following five variants, the last three of which only find one threshold per hidden unit:

- No discretisation besides the online selection performed by C4.5
- Activation clustering with a maximum of 10 clusters per neuron (and therefore 9 threshold)
- Simulated tree builder with a maximum of 1 threshold per neuron
- Activation clustering with a maximum of 2 clusters per neuron (and therefore 1 threshold)
- The medium of the activation range, which is zero as the hyperbolic tangent function is used

Finally, variants are considered for which the post-pruning approach (Section 4.1.3) is applied to the intermediate and final rules, as well as those where no such pruning takes place.

For the second part of the evaluation, only the discretisation variants which perform best according to the first comparison for the cases of selecting multiple and one thresholds per neuron are considered for each retraining and post-pruning variation. As outlined in Section 5.3, experiments are performed using different percentages of the data to inspect to what extent the expression can be reconstructed when less data is available. This results in performing 30 experiments per dataset per selected configuration. The performace of these models is compared with that built by the C4.5 approach, with and without appling the same post-pruning procedure.

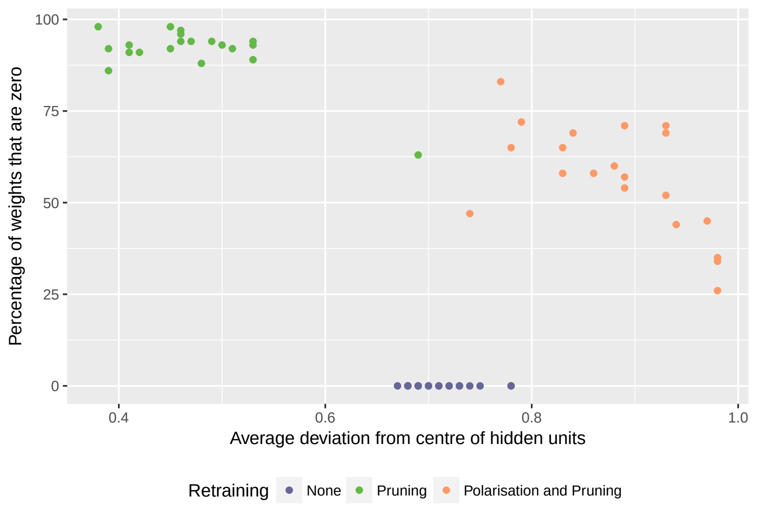


Figure 5.2.: Trade-off between the percentage of weight connections that can be equated to zero and the deviation from the activation centre.

5.6 Setting of the Hyperparameters

As considering different settings of the hyperparameters would have greatly increased the already elevated number of experiments per dataset, it was decided to select for each one only one setting which seemed appropriate; except for the upper bound on clusters when performing activation clustering, for which two settings are considered.

The implementation of C4.5 stops growing the tree when the percentage of instances which belong to the majority class exceeds a threshold or there are less instances that some percentage of the initial dataset size. In Zilke et al. (2016), a more thorough comparison of the effect of these parameters in the context of rule extraction from deep neural networks is performed, and it is shown that a higher class dominance threshold and a lower minimum dataset sise requirement correlate with the final expressions more strongly mimicking the network. Also, as the post-pruning approach is applied over the expressions extracted from the tree, it was decided to set these parameters at 99% class dominance and 1% minimum dataset size. Additionally, only binary splits are allowed and the trees have a maximum depth of twenty nodes.

For performing activation polarisation, ρ is set at 0.99, β is increased by 0.1 at a time and 1000 epochs are performed after each increase. The retraining is stopped when either a decrease in accuracy occurs or the divergance error (not multiplied by the weighting factor) falls below 0.01.

For weight sparseness pruning, a 1% decrease in accuracy is chosen, as this results in pruning a much greater percentage of connections than if no decrease is allowed, but not considerably less than when allowing a 2% decrease. Each time a connection is pruned, 1000 epochs are performed before considering the accuracy. For the networks which are retrained in both manners, the penalty term from the activation polarisation is added to the loss function used during connection pruning, multiplied by the last accepted value of β.

5.7 Characteristics of the Trained Networks

The possibility was recognised that pruning connections may concentrate more logic in the remaining neurons, thus having to observe a greater number of thresholds per neuron to make sense of the hidden unit's function.

After training all networks and performing the retraining steps, it was observed how many weight connections had been pruned, and how well the neurons could be polarised by calculating the average deviation of the hidden units from zero (the centre of the range for the hyperbolic tangent function) in the train data (5.2). These characteristics are depicted in Table 5.3.

$$\frac{\sum_{i\in D}\sum_{h_{l,n}\in N}\left|a^i_{l,n}\right|}{|D|\,.\,|N|},\quad N := \left\{h_{l,n} \mid l \in [1, \text{output layer})\right\} \tag{5.2}$$

The results of this analysis corroborate the hypothesis, and show that the effect is much more pronounced than was expected. A clear trade-off seems to take place between the divercence from the range centre and the number of connections that can be disregarded. This is illustrated in Figure 5.2, where the values are shown for the original networks,

Formula	**Hidden nodes**	**Avg. activation deviation from centre**			**Percentage zero weights**		
		original	pruned	polarised, pruned	original	pruned	polarised, pruned
$F1$	18-10-2	0.69	0.46	0.83	0	97	65
$F2$	20-11-2	0.72	0.53	0.98	0	94	35
$F3$	12-7-2	0.70	0.48	0.93	0	88	52
$F4$	20-11-2	0.71	0.49	0.89	0	94	71
$F5$	14-8-2	0.73	0.39	0.93	0	92	69
$F6$	12-7-2	0.73	0.41	0.89	0	91	57
$F7$	22-12-2	0.74	0.45	0.78	0	98	65
$F8$	16-9-2	0.67	0.50	0.84	0	93	69
$F9$	28-15-2	0.78	0.46	0.93	0	96	71
$F10$	14-8-2	0.69	0.45	0.97	0	92	45
$F11$	16-9-2	0.75	0.53	0.86	0	89	58
$F12$	16-9-2	0.68	0.51	0.98	0	92	26
$F13$	14-8-2	0.69	0.53	0.94	0	93	44
$F14$	14-8-2	0.71	0.47	0.83	0	94	58
$F15$	12-7-2	0.68	0.42	0.74	0	91	47
$F16$	18-10-2	0.68	0.46	0.79	0	94	72
$F17$	18-10-2	0.70	0.39	0.88	0	86	60
$F18$	14-8-2	0.72	0.41	0.89	0	93	54
$F19$	24-13-2	0.73	0.38	0.77	0	98	83
$F20$	16-9-2	0.78	0.69	0.98	0	63	34

Table 5.3.: Characteristics of the trained neural networks.

those retrained with weight sparseness pruning and those where activation polarisation was applied, followed by weight sparseness pruning.

The pruning approach was able to eliminate a great percentage of the connections, but at the cost of distributing the activation values more evenly across the range. When the networks were first polarised, and the penalty term was maintained in the cost function during the latter connection pruning, the activation values remained close to the range boundaries, but much less connections were eliminated.

6 Evaluation

In Chapter 4, several techniques were presented for altering the internal behaviour of trained neural networks and for performing offline discretisation on the activation ranges of the hidden units, as well as an approach for pruning rule sets. In the following sections it is explored how these modifications fare in terms of the complexity of the intermediate models and the semantic similarity of the extracted expressions to the original boolean formulas.

6.1 Reconstruction Using the Entire Dataset

Before regarding the generalization habilities of each variant, a quality assessment of the concepts reconstructed when the entire dataset is available is considered. A comparison is performed of the interpretability of the intermediate concepts, as well as of the accuracy on the data used to train the models.

Achieving a high accuracy on the train data is not usually considered as a primary goal, so the motivation for observing this may be unclear. However, in this context correctly predicting the class values for the train instances is not trivial. In fact, there are several reasons why a concept may not be correctly reconstructed even when all input combinations are available. For starters, each individual decision tree is likely to include some error. Trees are built in a greedy manner by sequentially selecting the split value that appears likelier to lead to the best solution, but it is possible that only a local minimum is found.

The process of growing a tree ends when one of the stopping criteria is reached. In the best case scenario this would occur when a high percentage of the instances in each node belong to the node's majority class, but there is also a chance that there is not enough data to continue branching, that the maximum defined depth is reached or that no remaining split value causes a positive information gain.

The last scenario is particularly likely when an offline discretisation has taken place, as the selected split values may not be sufficient to separate the data appropiately. However, letting the tree building process select any midpoint between two subsequent values may also lead to significant errors, as the tree could overfit the data. This may not seem problematic when the goal is to achieve a high training accuracy, but it is if overfitting occurs in a deep layer. For instance, consider the situation that the condition $h_{3,2} > 0.98$ is selected when building a tree between the third and fourth hidden layers, and that this condition is maintained after pruning because it improves the accuracy between the two specified layers. In the event that only a small percentage of the instances fall to one side of the condition, there may not be enough data to build the expression $DNF_{h_2 \rightarrow h_{3,2} > 0.98}$, and so by looking to maintain a high accuracy for the intermediate expressions, the accuracy of the final expression $DNF_{h_0 \rightarrow target}$ is damaged.

Considering this situation, the goals of achieving a high accuracy and building simple intermediate concepts may not be conflicting. If less and simpler intermediate expressions are constructed, it is likelier that these hold for a greater percentage of the data, and the chance for any one instance to fall within the error range of one of the intermediate concepts is lower. Therefore, in this section both these properties are analysed.

6.1.1 Effect of Applying Post-pruning

First, the effect of applying the post-pruning method on each intermediate expression and between substitution steps was observed. This was overwhelmingly positive, substantially reducing the complexity of the intermediate concepts and either having no effect or a positive one in the accuracy. More specifically, from the 293 configurations for which an expression was extracted with and without applying post-pruning, this step increased the accuracy in 123 cases and only decreased it in 7. The effect is illustrated in Figure 6.1. [1]

The positive effect in the accuracy would be a trivial finding if the post-pruning had been applied at the end in the final expression, as it is the accuracy on the train data which is being regarded. However, it should be kept in mind that by reducing the complexity of the intermediate concepts within some layer h_i these could be simplified too strongly, for instance disregarding certain neuron thresholds from neurons in layer h_{i-1} which are necessary to regain the pattern modeled by the input vectors.

Because of the consistency in the results, only those configurations are further compared which include the post-pruning.

[1] An overview of the experiments which were not completed is given in Section 6.3.

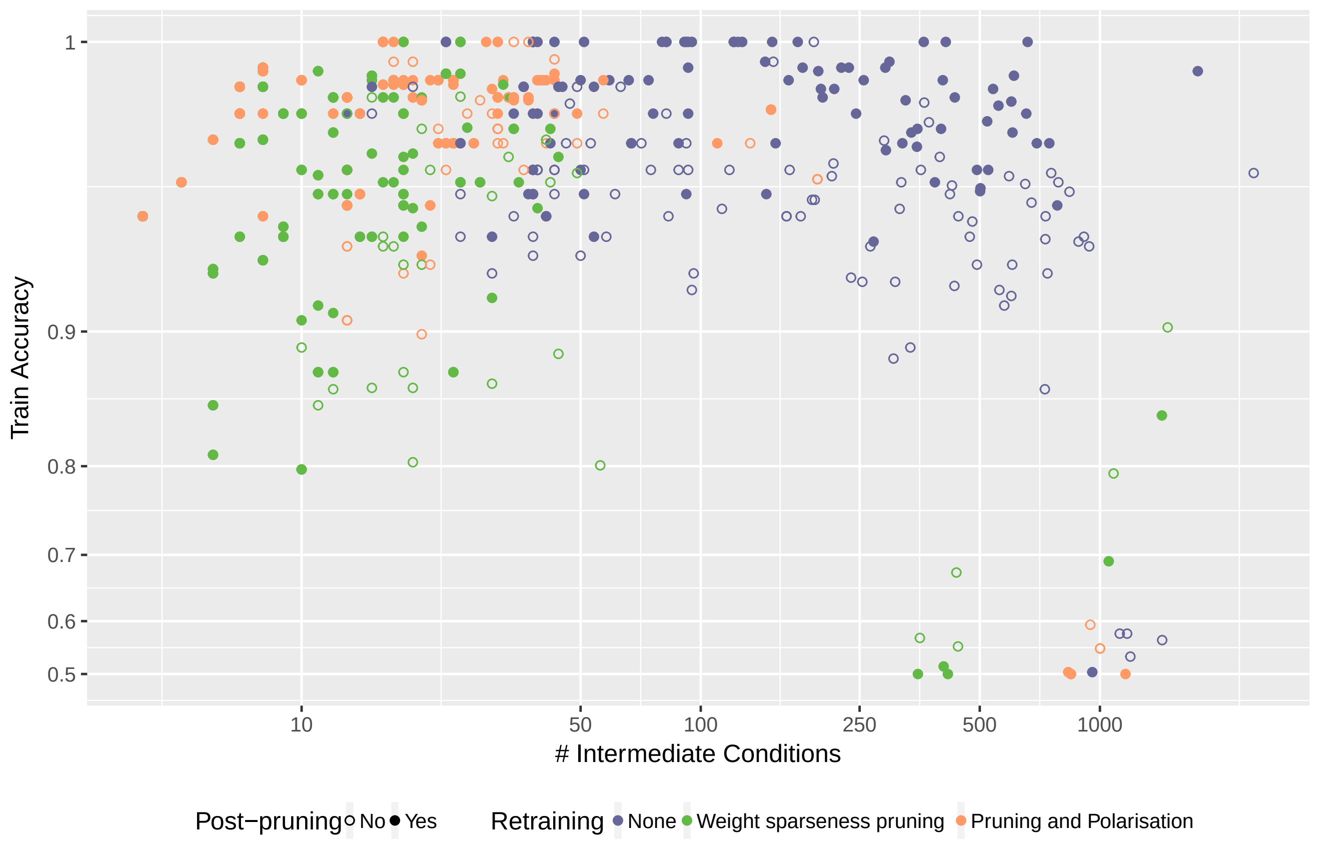


Figure 6.1.: Effect of applying the post-pruning approach and different retraining methods on the complexity of the intermediate concepts and the train accuracy of the final expressions.

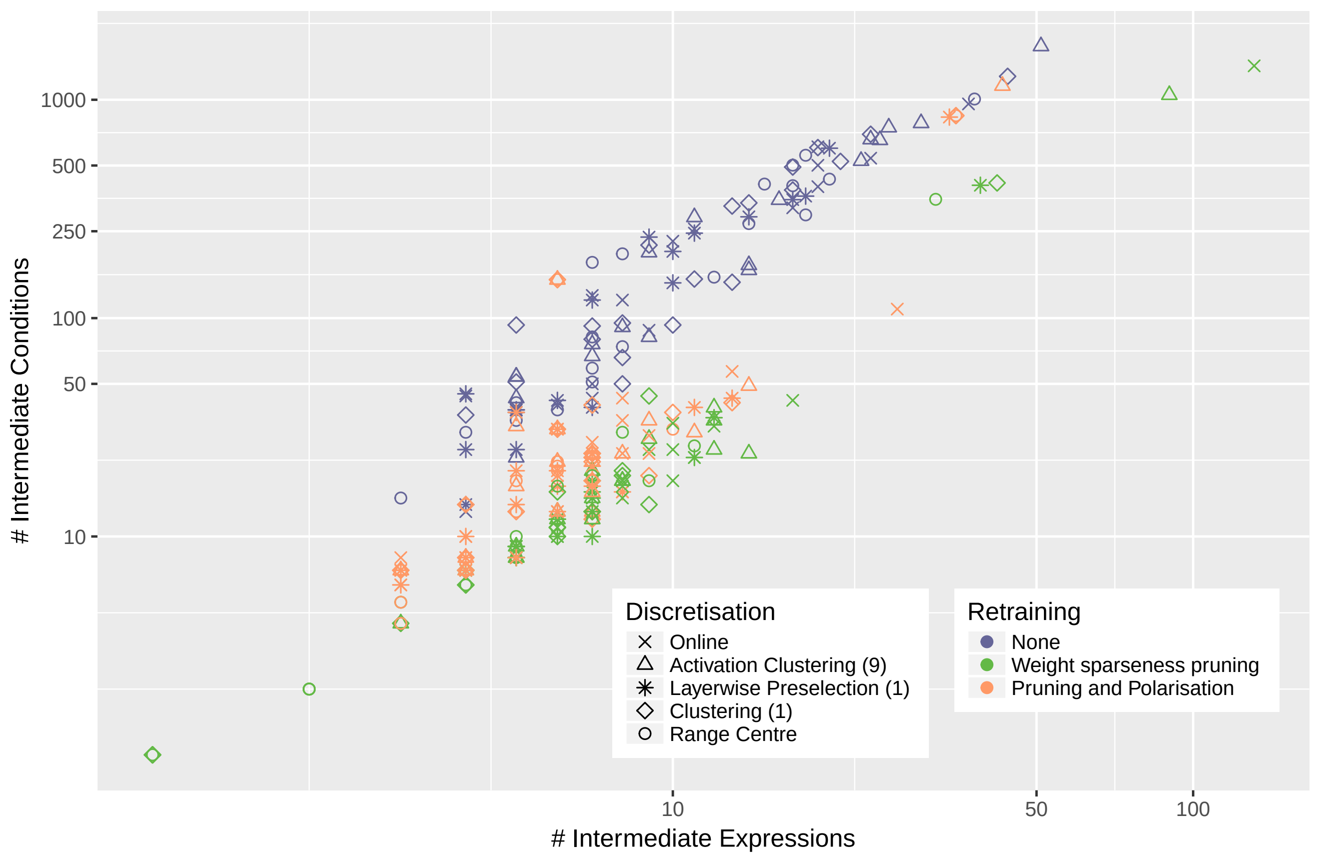


Figure 6.2.: Complexity of the intermediate concepts.

6.1.2 Complexity of the Intermediate Concepts

Applying the different retraining methods on the networks did not cause a significant change in accuracy when no discretisation was performed as almost all expressions could be reconstructed to at least 95%.

There was, however, a noticeable change in the number of intermediate expressions which were extracted – as well as in their complexity measured with the number of total conditions – when the networks were retrained under weight sparseness pruning. The models which resulted from less connected networks were much more compact.

This ocurred with and without a previous polarisation of the activations, although the total number of conditions was reduced to a greater degree when only the pruning was performed. Considering that a significantly lower number of weight entries were nulled in the polarised and pruned networks (Section 5.7), the effect is surprisingly similar.

The different discretisation approaches did not change the complexity of the intermediate expressions in a significant way for either of the complexity measures. However, applying activation clustering in some cases resulted in some models having a greater number of intermediate expressions and conditions than when no offline discretisation was carried out. The complexity of the models is illustrated in Figure 6.2.

6.1.3 Accuracy on the Train Data

Regarding the accuracy for the models extracted from unpruned networks, the results show only a significant difference, found as such by both tests, between the configuration which did not include a discretisation step and that where two clusters of activation values were formed on each neuron and the single midpoint of the boundaries was used to divide that range, which performed worse according to both tests (13 losses for N=18 [2], Z=2.09).

Interestingly, besides this particular case, there was no significant difference between any two discretisation approaches neither for the models which did not include any network retraining nor between those for which the networks were pruned and polarised; yet there was a change between any two discretisation methods when the network was pruned but not polarised. For this case, restricting the number of split values in any manner significantly decreased the accuracy,

[2] There was an aborted experiment for $F9$.

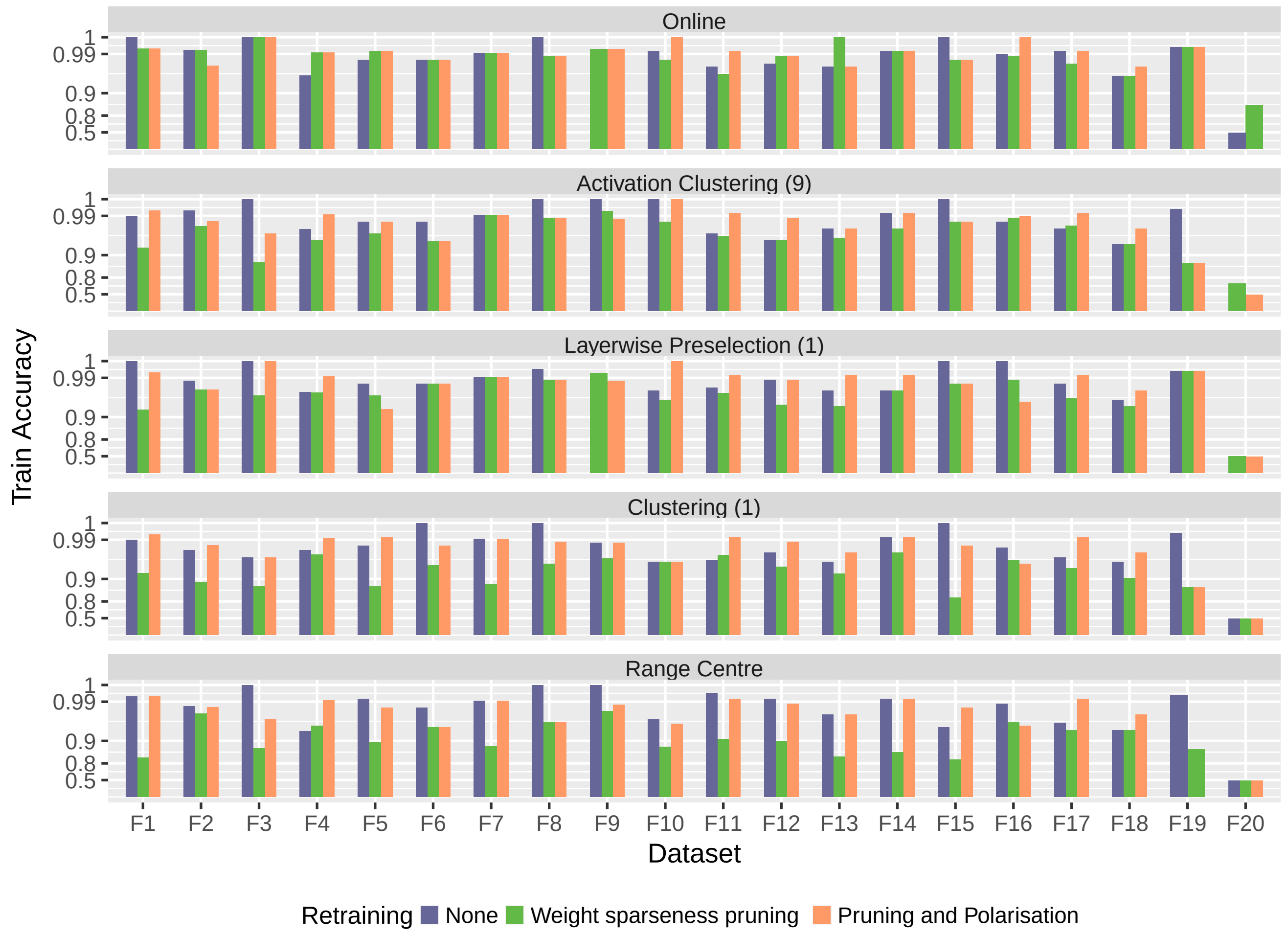


Figure 6.3.: Extent to which the concepts modeled by the neural networks could be reconstructed when using all input combinations. Post-pruning was applied when building all models.

as can be observed in Figure 6.3 by regarding the change in the green bars. The only exception where there was no significant difference was between the activation clustering with nine midpoints and the layerwise preselection of one split value per neuron.

These results reinforce the hypothesis that, when a high number of network connections are pruned and a retraining phase is performed between pruning steps, the logic modeled by the network is more heavily concentrated in the remaining neurons, thus needing to subdivide the neuron range into more intervals so as to allow each neuron to be described.

Figure 6.3 illustrates the extent to which the concepts modeled by the neural networks could be reconstructed when using all input combinations for two different network retraining methodologies and five ways of selecting where to partition the activation ranges, always applying post-pruning.

6.2 Reconstruction Using Part of the Dataset

After exploring the limits of each approach, the generalisation abilities of the different variants were compared by utilizing a subset of the combinations as train data and analyzing the accuracy on the remaining instances. A reduced number of discretisation variants was considered per configuration. As there was no significant difference neither in terms of the accuracy nor with regards to the comprehensibility of the intermediate concepts between performing activation clustering with nine thresholds per neuron and relying on the natural discretisation of C4.5, the second variant was considered.

For the case of using only one threshold per neuron, the layerwise threshold preselection approach was selected as it was the best performing amongst those which adhered to this constraint, particularly in the case when weight sparseness pruning was performed but no polarisation of the activations.

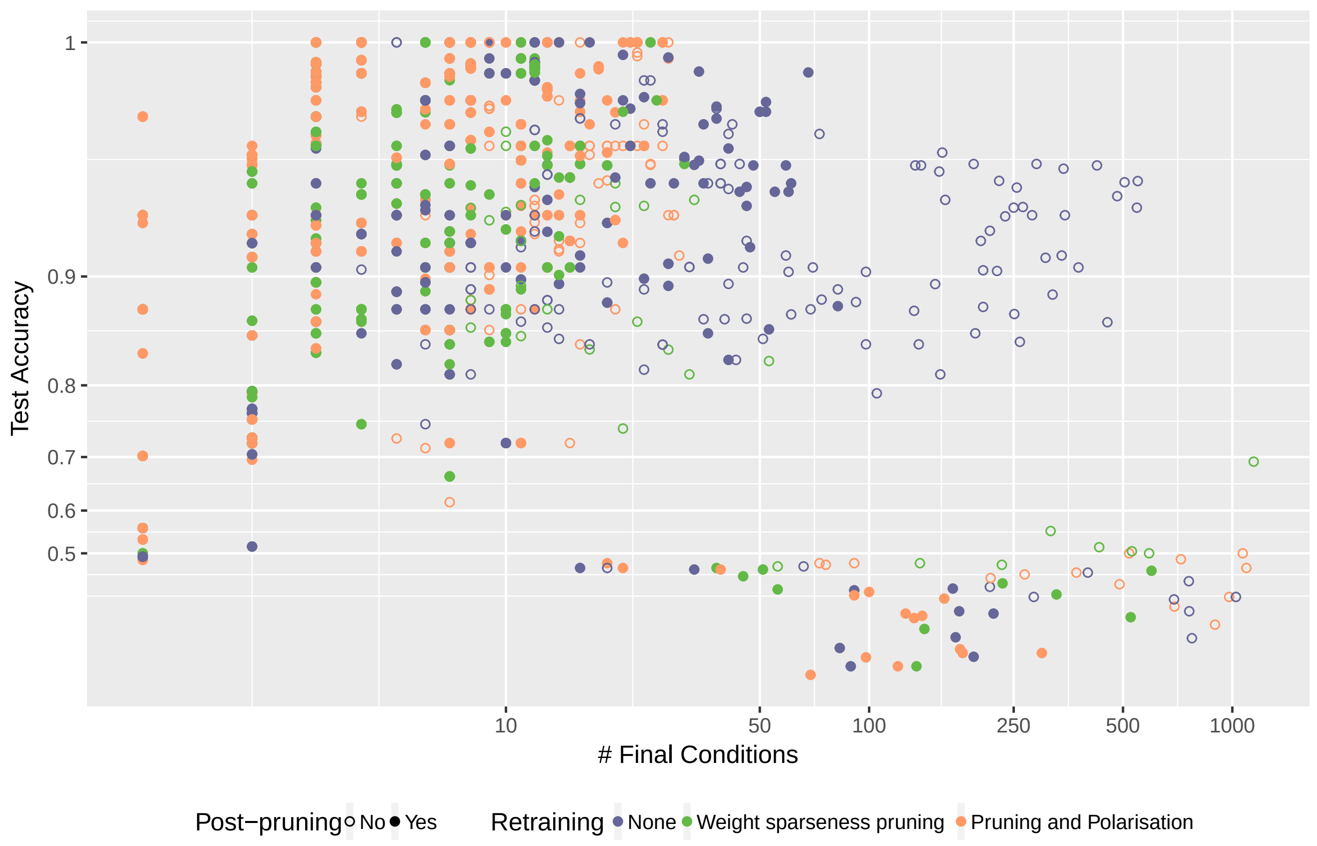


Figure 6.4.: Effect of applying the post-pruning approach and different retraining methods on the accuracy and complexity of the final expression.

Additionally, the configuration of considering zero as the only possible threshold was observed for the case where the network has gone through both retraining steps, as it is the simplest method and obtaining good results in this manner would considerably facilitate the rule extraction process.

6.2.1 Effect of Applying Post-pruning

The expectations for performing the post-pruning step were less certain in this context, as it may have contributed to overfitting the training data. However, this did not come to pass. From the 691 configurations for which models with and without applying post-pruning were present, the method increased the predictive accuracy 146 times and only caused a decrease 52 times. The results can be observed in Figure 6.4.

Considering this, further comparisons are only made between models built applying the post-pruning. As the effect was also highly positive for the C4.5 approach, increasing the accuracy 16 times and decreasing it only 6, it is this variant which is further compared to the decompositional approaches.

6.2.2 Accuracy on Unseen Instances

Generally, the best performing approaches when less data was available were those for which weight sparseness pruning as well as activation polarisation had been performed on the networks. Often, these methods were able to generalise better than the C4.5 variant. However, significant differences were only found in some cases.

When considering only the models extracted from networks which had not gone through any retraining, none of the tests found a significant difference between the different discretisation procedures. The same occurred for models extracted from networks that had been polarised and later pruned, even when only zero was used as a neuron threshold.

These results are encouraging in the sense that only considering a small number of split values makes the rule extraction process more applicable to very large datasets, so the fact that the accuracy was not negatively affected suggests rule representations could be extracted for models trained with big amounts of data without having to discard part of it. However, they speak against the expectation that restricting the number of thresholds would allow for a better generalisation.

Only for the cases when network pruning was performed but no activation polarisation took place was there a noticeable difference, as the models for which the activation ranges had only been discretised in an online manner by C4.5 outperformed the discretised variant according to the Wilcoxon test for the cases where 25% (Z=2.16) and 75% (Z=2.44) of the data was used. This is consistent with the significant decrease in train accuracy reported earlier (Section 6.1.3).

When observing the difference between the models built without performing any offline discretisation, which is to say when only regarding only the effect of the retraining practice, the variants which included weight sparseness pruning generally performed better than those which did not, regardless of whether the activations had been polarised. The pruned variant outperformed that where no retraining had taken place according to both tests when 25% of the data was used (with 16 from N=19 wins, Z=3.11), and that for which both retraining methods had been applied was deemed better by the Wilcoxon test when using 10% of the data (Z=2.07). Between both approaches that included pruning connections of the network, there was no significant difference for any data percentage.

It is worth noting that the inconsistency in the performance, that is to say in the fact that sometimes the unpruned approaches outperformed the pruned ones and other times the opposite occured, is partly due to the original decrease in accuracy of up to 1% that occurred when retraining the networks.

Finally, compared to the C4.5 approach which included post-pruning, none of the variants which did not perform network pruning showed a significant improvement. From those that did prune irrelevant connections and did not polarize the activations, only that for which no additional discretisation took place significantly outperformed C4.5 when 50% and 10% of the data was present according to the Wilcoxon test (with Z=2.22 and Z=2.63, respectively) and for both tests when 25% was used (16 wins out of N=20, Z=3.06).

The same situation held for the polarised and pruned variant (with, respectively, Z=2.31, Z=2.52, Z=2.75 and 14 wins out of N=19). However, this variant also outperformed C4.5 in several cases using only one threshold per neuron. When only the centre of the range was taken, C4.5 was worse according to both tests when 25% of the data was used (with 16 wins from N=19 and Z=3.40). The same held when using the layerwise preselection (with 16 wins from N=20 and Z=3.23), only here the Wilcoxon test also found a difference for the 10% split (with Z=2.17).

The general performance of the approaches for all datasets is illustrated in Figure 6.5. It is noticeable that when networks were retrained to reduce connectivity and encourage activation polarisation, the models almost always performed within the best and the discretisation did not have a great effect, whereas when the networks had not gone through any retraining the models performed similarly to those built by C4.5. This effect was inversely proportional to the data availability. A comparison per dataset of the retraining methods without using additional discretisation can be seen in Figure 6.6. The accuracy scores of the approaches when using 10%, 25% and 50% which were used for the comparison are found in the annex in Table A.1, Table A.2 and Table A.3, respectively.

6.3 Failed Experiments

There were two circumstances under which experiments did not build an acceptable model. The first was that either the memory or time constraints were surpassed, which were set at 24 hours and 5000 MegaBytes respectively.

The second situation occured when the first decision tree which would define a model for the target condition was not able to find any split value which increased the gain, so a random default rule was chosen as the final model and no intermediate concepts were extracted. Note that this could occur either in the last hidden layer or shallower ones if deeper layers merely modelled the identity function.

Within the first comparison, a total of 600 experiments were performed, composed of 30 per dataset. For the second comparison, where seven configurations were reviewed and models were built using different percentages of the data, 8400 experiments in total took place, reduced to 1400 events by averaging the scores of the corresponding folds. The failures are summarised in Table 6.1.

Models could not be built for a total of three datasets. The first was *F9*, which was the largest dataset used with 16384 total instances. Here, a total of four experiments were not completed when using 100% of the data. Also, sixteen experiments, resulting in four events, were not completed when using 90% of the data. The problematic configurations were the same as when 100% was used. All these experiments aborted during the extraction of intermediate expressions due to time constraints.

The second problematic dataset was *F19*, for which all experiments performed in networks that had gone though polarisation and connection pruning could not be completed when zero was used as the only threshold as no acceptable split value was found. This situation was somewhat unexpected as it was otherwise easier to extract representations using this form of discretisation when the activations had been polarised. However, it is worth noting that the model extracted from the network which had not been polarised but had had connections removed was also a default rule, and the intermediate concepts extracted were merely the identity function.

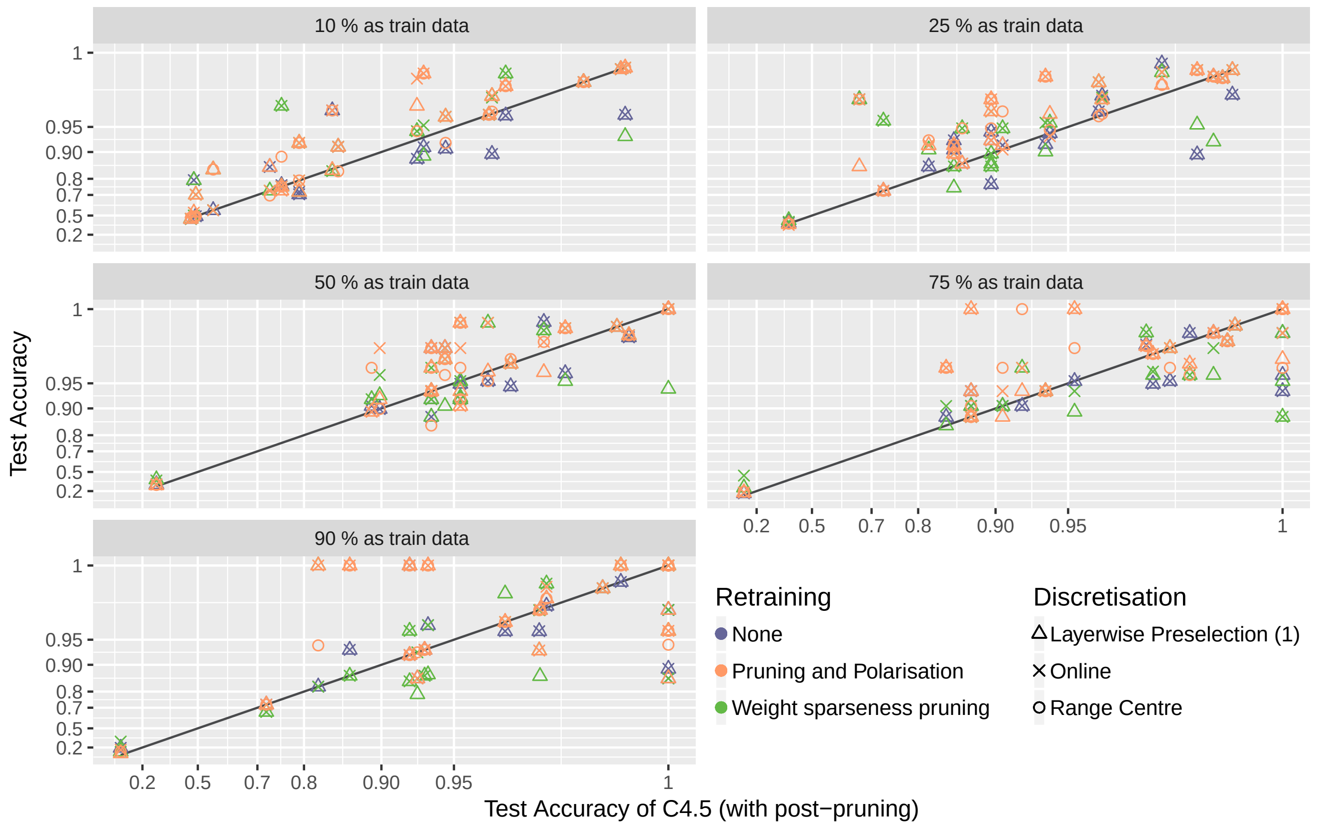


Figure 6.5.: Accuracy of the final expression for all datasets when using a subset of the data for building the model, on the remaining instances. Several variants of the approach, using different retraining and discretisation methods, are compared to the C4.5 algorithm. Post-pruning was applied when building all models.

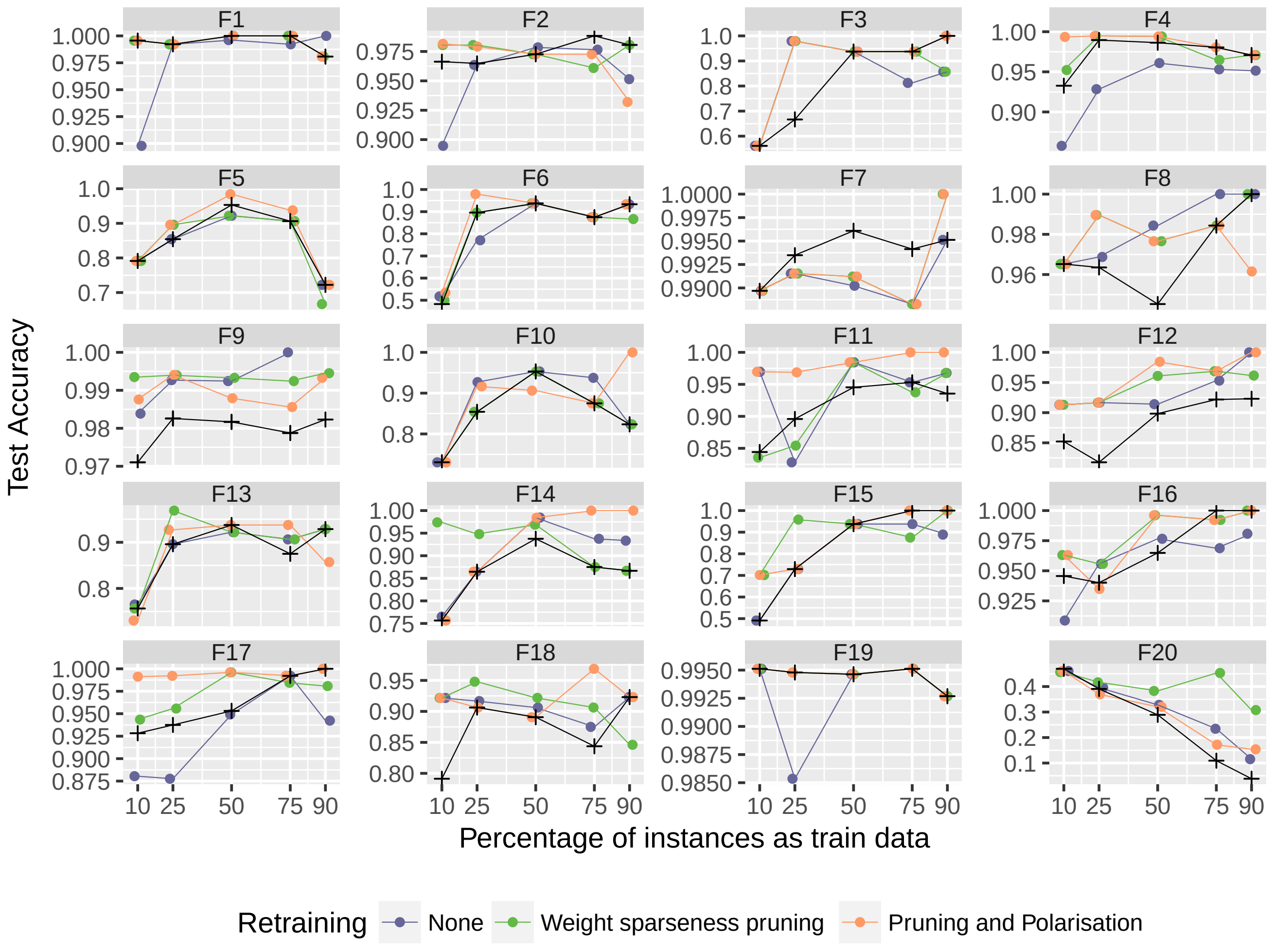


Figure 6.6.: Accuracy of the final expression per dataset when using a subset of the data for building the model, on the remaining instances. Different forms of retraining were applied to the networks prior to rule extraction. All models were built using the post-pruning approach and without applying any offline discretisation. A horizontal jitter of 0.1 was added.

Finally, the dataset *F20*, which models the parity function with eight inputs, caused experiments to abort due to exceeding the time limit during the substitution phase. For the case where 100% of the data was considered, this occured for a total of six configurations. Additionally, a total of four experiments aborted when a subset of the data was used, resulting in three further events not being completed.

Dataset	**Reason for Failure**	**Problematic Configurations**			
		Data %	Retraining	discretisation	Post-pruning
$F9$	Execution time exceeded limit	100, 90	none	none, layerwise preselection (1)	no, yes
$F19$	No split value found by the single decision tree in a layer	all	both	range centre	no, yes
$F20$	Execution time exceeded limit	100	none	activation clustering (9)	no
		100	none	activation clustering (1)	no
		100	both	none	no
		100	none	activation clustering (9)	yes
		100	none	layerwise preselection (1)	yes
		100	both	none	yes
		75	pruning	none	no
		50	pruning	none	no
		90	both	none	no

Table 6.1.: Summary of the configurations under which no model was constructed. Retraining the network in both manners refers to polarizing the activations and then pruning connections.

6.4 The Case of Reconstructing the Parity Function

The parity function with eight inputs, referred to as expression *F20*, paints a rather different picture as the rest of the expressions. When the entire dataset was used for rule extraction, several of the experiments aborted due to time constraints during the substitution phase, as too large intermediate expressions were build. Those models which could be constructed also built significantly more complex intermediate concepts than those extracted for the other functions, as can be identified in Figure 6.1 at the bottom of the y axis and right side of the x axis.

Interestingly, the average accuracy of the intermediate expressions per layer for deeper layers was over 0.98 for all hidden layers except the first. This number falls to 0.928 for the expressions extracted between the input and the first hidden layer, where the majority of the logic is concentrated. As the mistakes were spread widely across the instance space, when the final expression was merged they accumulated.

It is worth mentioning that when C4.5 is applied directly on the entire dataset which models the parity, the default rule is returned, as no single test partitions the instance space into subsets where the classes are more evenly distributed. A similar phenomenon occurs when building decision trees between the input and the first hidden layer.

This is best illustrated by the tree built for $h_{1,1} \leq 0$ in the diagram of the general workflow in Figure 4.1, which would actually not be able to be constructed, as C4.5 would need to perform both tests $x_0 \leq 0.5$ and $x_1 \leq 0.5$ in order to obtain a positive information gain. Of course, it is highly unlikely that this would occur in a real setting, or even if more attributes were present, as even if a similar pattern were to underline the data, at least one of the possible split values that would be considered by C4.5 would likely result in a positive, albeit perhaps small, information gain.

For the experiments which successfully terminated, the expression could only be reconstructed to an acceptable degree by two configurations which include weight sparseness pruning, namely those where no additional discretisation is performed, which reached an accuracy of 0.902 when no post-pruning was applied, and that where activation clustering was used, where the accuracy was at 0.801. In both these cases, the average accuracy of expressions extracted between the first hidden layer and the inputs was equal to one, as any neuron of the first hidden layer was only connected to a few of the inputs, and therefore the standstill characteristic of the parity function was never reached. It should also be noted that the parity function was one of the only datasets where applying post-pruning on the intermediate expressions and between substitution steps caused a fall in accuracy. The dataset is easy to recognise in Figure 6.1 and Figure 6.4 due to the considerably lower accuracy.

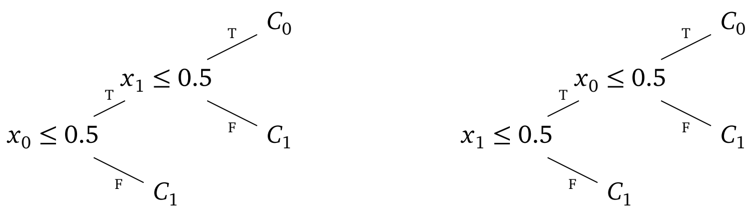


Figure 6.7.: The two decision trees C4.5 may build from the information $x_0 = 0, x_1 = 0$ (C_0), $x_0 = 0, x_1 = 1$ (C_1) and $x_0 = 1, x_1 = 0$ (C_1). Both trees would classify $x_0 = 1, x_1 = 1$ as C_1.

When looking to reconstruct the parity function with a subset of the data, another intriguing phenomenon occurred. Whereas for the other datasets increasing the amount of training examples helped classify the unseen instances, the opposite happened for the models built for the parity function, where the predictive accuracy decreased as more instances were made available.

The same happens for C4.5. Whereas if all the data is available no split value leads to a positive gain and the process terminates, when a reduced amount is present a model can be built, but it does not generalise to the rest of the data.

For instance, consider the situation of reconstructing the parity function with two inputs when the observations $x_0 = 0, x_1 = 0$ (C_0), $x_0 = 0, x_1 = 1$ (C_1) and $x_0 = 1, x_1 = 0$ (C_1) are used. In that case one of the trees shown in Figure 6.7 would be built, both of which incorrectly classify the instance $x_0 = 1, x_1 = 1$ as C_1.

To observe whether the same would hold for deep neural networks, which more easily model nonlinear relationships, a different network was trained for each of the folds that was used for rule extraction. Table 6.2 shows the accuracy of the models, averaged over all folds corresponding to the same data percentage. The results show that the networks are naturally drawn to modelling the parity as the number of input combinations grows. This property was maintained when the networks went through connection pruning, although in these cases the accuracy slightly decreased instead of improving from simplifying the models.

	10%	**25%**	**50%**	**75%**	**90%**
Original network	0.443	0.426	0.459	0.694	0.923
Network after polarisation and connection pruning	0.410	0.421	0.472	0.679	0.882
Network after connection pruning	0.425	0.438	0.433	0.652	0.901

Table 6.2.: Predictive accuracy of neural networks trained with different percentages of the dataset corresponding to the parity function with eight inputs. The accuracy scores are averaged over all folds corresponding to the same data percentage.

The difficulty in extracting hidden features that neural networks seem to construct naturally displays a considerable weakness of the rule extraction approaches.

6.5 Interpreting Hidden Concepts

Although the number of intermediate expressions and the amount of conditions within these are useful measures to assess the comprehensibility of models describing the internal structure of neural networks, they do not illustrate directly how it would be to interpret such models.

In this section the case is considered of interpreting the intermediate expressions extracted from reconstructing a concept which was regained correctly by several approaches, yet for which the extracted hidden concepts varied greatly, namely the formula:

$$F3 : (K \wedge Q \wedge T) \vee \left(\bar{K} \wedge \bar{M} \wedge \bar{P} \wedge R \wedge \bar{T}\right)$$

The mapping of the literals to the inputs of the networks is defined as:

$$M \rightarrow x_0 \quad Q \rightarrow x_1 \quad R \rightarrow x_2 \quad K \rightarrow x_3 \quad P \rightarrow x_4 \quad T \rightarrow x_5$$

The final expression after reconstruction thus has the following form:

$$DNF_{h_0 \to target} : (x_3 \wedge x_1 \wedge x_5) \vee \left(\bar{x}_3 \wedge \bar{x}_0 \wedge \bar{x}_4 \wedge x_2 \wedge \bar{x}_5\right)$$

The expression may be built from an unlimited number of intermediate concepts. Here, two sets of intermediate expressions which allow the reconstruction of the specified expression are observed. They were both built by approaches which did not perform offline discretisation, namely the variant which did not retrain the network and that which polarised the activations and pruned the network and prunes irrelevant connections.[3]

When the network was not subjected to any retraining, the following intermediate expressions were extracted:

$$\begin{aligned}
&DNF_{h_3 \to target} : h_{3,1} \leq 0.15\\
&DNF_{h_2 \to h_{3,1} \leq 0.15} : h_{2,6} > 0.32 \vee h_{2,4} > 0.56\\
&DNF_{h_1 \to h_{2,4} > 0.56} : \left(h_{1,0} \leq -0.26 \wedge h_{1,7} \leq -0.65\right) \vee \left(h_{1,0} \leq -0.26 \wedge h_{1,8} > 0.12\right)\\
&DNF_{h_1 \to h_{2,6} > 0.32} : h_{1,6} > 0.32\\
&DNF_{h_0 \to h_{1,8} > 0.12} : \left(x_4 \wedge x_5\right) \vee (x_1 \wedge \bar{x}_2 \wedge x_5) \vee \left(x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \wedge x_4\right) \vee (x_1 \wedge \bar{x}_3 \wedge x_5)\\
&DNF_{h_0 \to h_{1,7} \leq -0.65} : (\bar{x}_0 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_5) \vee \left(\bar{x}_1 \wedge x_2 \wedge x_4 \wedge x_5\right) \vee (\bar{x}_0 \wedge x_2 \wedge \bar{x}_3) \vee \left(x_3 \wedge x_4 \wedge x_5\right)\\
&DNF_{h_0 \to h_{1,0} \leq -0.26} : x_1 \wedge x_3\\
&DNF_{h_0 \to h_{1,6} > 0.32} : \bar{x}_0 \wedge x_2 \wedge \bar{x}_3 \wedge \bar{x}_4 \wedge \bar{x}_5
\end{aligned}$$

The model finds an adequate representation for the second conjunction $\left(\bar{x}_3 \wedge \bar{x}_0 \wedge \bar{x}_4 \wedge x_2 \wedge \bar{x}_5\right)$ in $h_{1,6} > 0.32$, as well as for the first part of the first conjunction $(x_1 \wedge x_3)$ in $h_{1,0} \leq -0.26$. But instead of finding a simple rule for x_5 which would finish the second conjunction, it relies in the two expressions extracted for $h_{1,7} \leq -0.65$ and $h_{1,8} > 0.12$, which include several conjunctions which cannot be joined with $h_{1,0} \leq -0.26$ because they include the negation of x_1 or x_3.

For the case where extraction was preceded by retraining the network, the following expressions were obtained:

$$\begin{aligned}
&DNF_{h_3 \to target} : h_{3,1} \leq -0.10\\
&DNF_{h_2 \to h_{3,1} \leq -0.10} : h_{2,6} > 0.88 \vee h_{2,6} \leq -0.99\\
&DNF_{h_1 \to h_{2,6} \leq -0.99} : h_{1,0} \leq -0.01 \wedge h_{1,8} > -0.01\\
&DNF_{h_1 \to h_{2,6} > 0.88} : h_{1,5} > -0.01 \wedge h_{1,8} \leq -0.01 \wedge h_{1,9} > -0.93\\
&DNF_{h_0 \to h_{1,9} > -0.93} : \bar{x}_0 \wedge \bar{x}_4\\
&DNF_{h_0 \to h_{1,0} \leq -0.01} : x_1 \wedge x_3\\
&DNF_{h_0 \to h_{1,8} \leq -0.01} : \bar{x}_5\\
&DNF_{h_0 \to h_{1,8} > -0.01} : x_5\\
&DNF_{h_0 \to h_{1,5} > -0.01} : x_2 \wedge \bar{x}_3
\end{aligned}$$

This model builds the first conjunction out of two expressions from the first hidden layer, and the second conjunction out of three expressions. A noticeable difference between the two representations is that here the expressions of the first hidden layer with regards to the inputs only include a few literals each, which makes it overall much simpler to trace the model. This characteristic is one of the reasons for which rule sets extracted from minimally connected networks generalise better.

6.6 Final Remarks

On some aspects, the results were highly consistent and provided a clear picture of what methods assist rule extraction from deep neural networks. However, other questions would require further information to be answered.

Reducing the connectivity of the network proved to be a robust way to extract simpler intermediate concepts which were better at classifying unseen instances. Yet it seems that encouraging low connectivity not only identifies irrelevant

[3] The original threshold values has eight decimal units. These were reduced to two to improve the visualisation.

logic created from training too-large architectures, but also concentrates the hidden features which are relevant for the classification into less neurons, thus requiring a more fine-grained partitioning of the activation ranges to regain the hidden patterns. This was partly shown by the analysis of the characteristics of the network (Section 5.7), and its negative consequences for rule extraction were confirmed by the dismal performance of models which combined connection pruning with putting an upper bound on the number of thresholds per neuron.

However, when polarisation of the activations was done jointly with connection pruning, the benefits of the latter could be leveraged while avoiding the undesired effect of concentrating more logic into less neurons. Although the number of connections which could be pruned in these networks was substantially lower, the intermediate models were not significantly more complex, and in terms of accuracy these approaches consistently performed within the highest.

Also, combining both methods for retraining was highly compatible with all offline discretisation techniques. In fact, usually very accurate models could be extracted even when only the centre of the neurons' range was considered, although this did not hold for the *F19* dataset.

Regarding the offline discretisation, the results were less consistent. One reason for exploring such methods was that when expressive hidden features are to be extracted, considering a reduced number of states per neuron gives a clearer picture of that neuron's role in the network. In fact, most decompositional approaches for extracting rules from neural networks only consider two states per hidden neuron, namely it being *active* or *inactive* (Section 3.3). The results show that it is in fact possible to restrict the number of thresholds per neuron and still obtain accurate representations. In particular the layerwise threshold preselection was proven capable of extracting accurate rule representations when only selecting one threshold per neuron.

Another setting where performing an offline discretisation may be desirable is when dealing with very large datasets, as the runtime of extracting the intermediate expressions increases considerably when the heuristic is calculated for a high number of split values. In fact, the extraction of intermediate concepts from the *F9* dataset was terminated for exceeding the time limit although at 16384 instances it is considerably smaller than some datasets used to train deep models. If the rule extraction process were to be applied to such models, then some form of offline discretisation would greatly accelerate the process.

In such a context, the layerwise threshold preselection would not be well suited, as the thresholds are chosen by a simulated tree building procedure which can require a similar amount of resources than building the separate trees. Performing activation clustering would, however, be a good alternative. The accuracy when applying this method did not significantly decrease and the rule extraction process was considerably faster. Also, within this work only the settings with ten and two clusters were observed, and a sufficiently high initial number of clusters would prevent the decrease in accuracy (Setiono, 1996). An additional advantage of this method is that it gives an assessment of the importance of each neuron before rule extraction takes place.

Finally, two last expectations of performing offline discretisation were that this would improve the generalisation capabilities of the approach and that the complexity of the intermediate concepts would decrease. However, none of these came to pass. In terms of accuracy, even the best performing configurations which used some discretisation were at best at the same level as their counterpart which partitioned ranges in an online manner, and there was no significant change in the complexity of the intermediate concepts.

A last note on the topic of offline discretisation is that although within this work all input values were boolean, in real life scenarios most types of data include continuous attributes, and activation clustering is an effective way of assessing the effect of discretising them in the network's accuracy.

7 Future Work

In most cases, the concepts learned by the networks could be reconstructed to a great extent and the complexity of the extracted hidden patterns was considerably reduced by reducing the connectivity of the networks. However, certain aspects should be addressed before representations obtained in this manner justify using deep models within safety critical applications.

One point is that by not explaining the network as a whole but only extracting the logic that seems the most relevant for the classification, the possibility is not excluded that some features are learned somewhere in the network that do not considerably affect the predictive accuracy but may cause the system to classify some instance in an undesirable manner.

Also, particularly the low performance when extracting the parity function shows that some improvements could be made in the approach in order to extract more reliable rules.

7.1 Augmentation of the Training Set

One way to increase the assurance that the network will react to new observations according to the obtained explanation is to include some degree of sampling within the extraction process.

This may of course result in much more complex models being extracted which include irrelevant characteristics. One possibility to reduce this effect would be to only use the training data to determine what split values are relevant within layer h_{l-1} for explaining conditions in layer h_l, but then sample all combinations of the activation of neurons in h_{l-1} being below or above the selected thresholds.

Augmenting the training set can, however, be problematic, as found by Milaré et al. (2001), as the network may react to implausible instances differently than to naturally occurring ones. Therefore, only activation combinations of layer h_{l-1} should be considered which result from likely feature vectors. One way to assure this would be to determine the range of each input feature and assess which feature combinations are unlikely to occur naturally. Then, the *Validity Interval Analysis* technique (Section 3.3.3) could be used to corroborate that the constraints are met.

7.2 Extraction of M-of-N Concepts

If for the problem at hand it is crucial to extract representations which consider all areas of the network but are not overly complex, M-of-N rules may be better suited than conjunctive ones. Such concepts could be extracted by using M-of-N conditions in the tree nodes as done by the *TREPAN* algorithm (Section 3.3.4).

However, to what extent it may be possible to group hidden units into equivalence classes without jeopardizing the accuracy would greatly depend on the network. It is possible that retraining the networks in some manner, as performed within the *MofN* algorithm (Section 3.3.2), may encourage such a property, but what effect this may have on the final models is yet to be explored.

7.3 Guiding Concept Learning

In the context of this work, networks were trained with randomly constructed boolean concepts and it was observed to what extent these could be regained. However, the problem formulation of providing an explanation for the path a deep neural network takes to make its classifications does not directly address the issue of increasing the applicability of the models in critical domains, but rather only presents a form of validation that some unwanted patterns are not learned.

In the last few years, neural-symbolic approaches have been developed to include domain knowledge in the form of first-order logical rules into the training of neural networks, thus providing a greater level of control over the learned features (Bader and Hitzler, 2005; d'Avila Garcez et al., 2009).

For instance, in Hu et al. (2016), a *teacher* network is maintained which projects a *student* network with the current parameters into a space which includes regularizing terms in the form or first-order rules. This networks aims to find the output activations which more closely resemble those of the student in terms of KL-divergence while adhering to the logic constraints. Both networks output probabilities for each class value, and the loss for an epoch which is used for updating the parameters is calculated using the outputs from both networks.

Considering this research, it would be interesting to train networks modeling real dataset while including such constraints, and then comparing the extracted expressions with the original logic rules, similarly to the analysis done by Towell and Shavlik (1993) but for larger architectures. Such a combined process would allow users to define which characteristics should be learned by the network, and then confirm that the guidelines have been adhered to.

8 Conclusion

In this work, deep neural networks were trained to emulate boolean formulas and the problem was explored of regaining these patterns while providing expressive insight into how the concepts were modeled within the networks.

Chapter 2 described some fundamentals on classifications problems and how these are tacked by decision tree approaches, as well as by artificial neural networks. The basic way was outlined in which the models are trained, as well as how they are later used for classifying new observations.

In Chapter 3, an overview was given of the research regarding the extraction of rule representations from neural networks, as well as that related to reducing network connectivity and performing offline discretisation of attribute ranges.

The methodology used to reconstruct the concepts from the neural networks was described in Chapter 4. Two different ways to retrain a deep neural network in order to assist latter rule extraction were explored, which resemble approaches that have been shown to help extract better rule representations from shallow networks. The first consists on pruning connections between neurons which are irrelevant for the classification, thus considerably simplifying the relations between those that remain. The second method looks to polarise the activation ranges of the hidden units so their state can be reduced to being *active* or *inactive*.

For extracting the intermediate and final rules, an existing decompositional approach was extended with a post-pruning step meant to lessen error propagation between layers and reduce the complexity of the model.

Also, it was explored how several offline discretisation techniques could be leveraged to set an upper bound on the number of activation thresholds that are considered per hidden unit. The first, which has been extensively used in the context of extracting rules form neural networks, consists of clustering the activation values of each neuron so that the accuracy of the network when the activations are replaced be their closest cluster mean remains acceptable. The second technique is inspired by an offline discretisation approach and involves choosing the thresholds for neurons of some layer which best partition the data according to all relevant thresholds of the adjacent deeper layer.

In Chapter 5, some information was provided regarding the selected boolean functions, as well as on the process according to which they were generated. Also, the trained neural networks were examined and it was shown that there is a trade-off between the extent to which connections can be pruned and the degree of polarisation the hidden units can achieve.

The results of the evaluation were described in Chapter 6. There, the importance of applying some post-pruning in the rule sets was clearly seen, as this proved to be an effective way of lowering the complexity of the concepts while increasing the accuracy.

Also, it was shown that pruning network connections markedly reduced the complexity of the intermediate expressions, allowing a much better analysis of the hidden features. This step also considerably improved the rate to which the patterns could be reconstructed when less data was available.

Overall, the best results were obtained when the activations of the networks had been polarised and irrelevant connections eliminated before extracting rules. In these cases, the discretisation of the activation ranges did not show a significant decrease in accuracy, even when the simplest discretisation was performed, namely taking the midpoint of the activation function's range.

Finally, in Chapter 7 some possible areas of future research were mentioned. Some alternatives were presented for addressing the weaknesses of the rule extraction process and it was described how rule extraction could work in junction with methods that directly guide what features are learned.

Bibliography

Igor Aizenberg, Naum N. Aizenberg, and Joos P. Vandewalle. *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Springer Science & Business Media, 2013.

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.

Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration - A structured survey. In *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 167–194. College Publications, 2005.

José M. Benítez, Juan L. Castro, and Ignacio Requena. Are artificial neural networks black boxes? *Institute of Electrical and Electronics Engineers Transactions on Neural Networks and Learning Systems*, 8(5):1156–1164, 1997.

Petr Berka and Ivan Bruha. Discretization and grouping: Preprocessing steps for data mining. In *Principles of Data Mining and Practice of Knowledge Discovery in Databases, Second European Symposium, Nantes, France, Proceedings*, pages 239–245, 1998.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *Computing Research Repository*, abs/1604.07316, 2016.

Robert K. Brayton, Gary D. Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

Thomas Burri. Machine learning and the law: Five theses. In *Proceedings of the Neural Information Processing Systems, NIPS 2016, Workshop on Machine Learning and the Law*, 2016.

Jason Catlett. On changing continuous attributes into ordered discrete attributes. In *Machine Learning - European Working Session on Learning, Porto, Portugal, Proceedings*, pages 164–178, 1991.

Michal R. Chmielewski and Jerzy W. Grzymala-Busse. Global discretization of continuous attributes as preprocessing for machine learning. *International Journal of Approximate Reasoning*, 15(4):319–331, 1996.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada, Proceedings*, pages 3123–3131, 2015.

Mark Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA*, pages 37–45. Morgan Kaufmann, 1994.

Mark Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems 8, Denver,CO, Proceedings*, pages 24–30, 1995.

Abhishek Das, Harsh Agrawal, Larry Zitnick, Devi Parikh, and Dhruv Batra. Human attention in visual question answering: Do humans and deep networks look at the same regions? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 932–937, 2016.

Artur S. d'Avila Garcez, Luís C. Lamb, and Dov M. Gabbay. Neural-symbolic learning systems. *Neural-Symbolic Cognitive Reasoning*, pages 35–54, 2009.

Janez Demšar and Dale Schuurmans. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Twelfth International Conference on Machine Learning, Tahoe City, California, USA, Proceedings*, pages 194–202, 1995.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341:3, 2009.

U. Mohammad Fayyad. *On the induction of decision trees for multiple concept learning*. PhD thesis, University of Michigan, 1992.

Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th International Joint Conference on Artificial Intelligence. Chambéry, France, Proceedings*, pages 1022–1029, 1993.

Douglas H. Fisher and Kathleen B. McKusick. An empirical comparison of ID3 and back-propagation. In *1th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, Proceedings*, pages 788–793. Morgan Kaufmann, 1989.

Alex A. Freitas. Comprehensible classification models: a position paper. *Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining*, 15(1):1–10, 2013.

LiMin Fu. Rule learning by searching on adapted nets. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA*, volume 2, pages 590–595, 1991.

Koji Fujimoto and Sampei Nakabayashi. Applying gmdh algorithm to extract rules from examples. *Systems Analysis Modelling Simulation*, 43(10):1311–1319, 2003.

Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *Stat*, 1050:31, 2016.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada, Proceedings*, pages 1135–1143, 2015.

Babak Hassibi, David G. Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems Conference 5, Denver, Colorado, USA*, pages 164–164, 1993.

Torsten Hothorn, Kurt Hornik, Mark A. van de Wiel, Henric Winell, and Achim Zeileis. *Conditional Inference Procedures in a Permutation Test Framework*, 2016. URL `http://coin.r-forge.r-project.org/`. R package version 0.5-20130417.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *54th Annual Meeting of the Association for Computational Linguistics, ACL, Berlin, Germany, Proceedings*, volume 1: Long Papers, 2016.

Samuel H. Huang and Hao Xing. Extract intelligible and concise fuzzy rules from neural networks. *Fuzzy Sets and Systems*, 132(2):233–243, 2002.

John Peter Jesan and Donald M. Lauro. Human brain and neural network behavior: a comparison. *Ubiquity*, 2003 (November):2, 2003.

S. M. Kamruzzaman and Ahmed R. Hasan. Rule extraction using artificial neural networks. *Computing Research Repository*, abs/1009.4984, 2010.

Nikola K. Kasabov. Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. *Fuzzy sets and Systems*, 82(2):135–149, 1996.

Ujwal Kayande, Arnaud De Bruyn, Gary L. Lilien, Arvind Rangaswamy, and Gerrit H. van Bruggen. How incorporating feedback mechanisms in a DSS affects DSS evaluations. *Information Systems Research*, 20(4):527–546, 2009.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computing Research Repository*, abs/1412.6980, 2014.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA]*, pages 598–605, 1989.

Zachary C. Lipton. The mythos of model interpretability. *Computing Research Repository*, abs/1606.03490, 2016.

Jian Liu and Ming Li. Finding cancer biomarkers from mass spectrometry data by decision lists. *Journal of Computational Biology*, 12(7):971–979, 2005.

Stuart P. Lloyd. Least squares quantization in PCM. *Institute of Electrical and Electronics Engineers Transactions Information Theory*, 28(2):129–136, 1982.

Dmitry M. Malioutov and Kush R. Varshney. Exact rule learning via boolean compressed sensing. In *Proceedings of the 30th International Conference on Machine Learning, ICML, Atlanta, GA, USA, Proceedings*, pages 765–773, 2013.

Claudia R. Milaré, ACPLF de Carvalho, and Maria C. Monard. Extracting rules from neural networks using symbolic algorithms: preliminary results. In *Computational Intelligence and Multimedia Applications, 2001*, pages 384–388. Institute of Electrical and Electronics Engineers, 2001.

Riccardo Miotto, Li Li, Brian A Kidd, and Joel T Dudley. Deep patient: An unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports*, 6, 2016.

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72:1–19, 2011.

Julian D. Olden and Donald A. Jackson. Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1):135–150, 2002.

J. Ross Quinlan. Generating production rules from decision trees. In *10th International Joint Conference on Artificial Intelligence. Milan, Italy, Proceedings*, pages 304–307, 1987a.

J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987b.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Raul T. Santos, Júlio C. Nievola, and Alex A. Freitas. Extracting comprehensible rules from neural networks via genetic algorithms. In *Combinations of Evolutionary Computation and Neural Networks, 2000 Institute of Electrical and Electronics Engineers Symposium on*, pages 130–139. Institute of Electrical and Electronics Engineers, 2000.

Makoto Sato and Hiroshi Tsukimoto. Rule extraction from neural networks via decision tree induction. In *International Joint Conference on Neural Networks, Proceedings*, volume 3, pages 1870–1875. Institute of Electrical and Electronics Engineers, 2001.

Rudy Setiono. Extracting rules from pruned neural networks for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 8(1):37–51, 1996.

Rudy Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1): 205–225, 1997a.

Rudy Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997b.

Rudy Setiono. Extracting m-of-n rules from trained neural networks. *Institute of Electrical and Electronics Engineers Transactions on Neural Networks and Learning Systems*, 11(2):512–519, 2000.

Rudy Setiono and Huan Liu. Symbolic representation of neural networks. *Institute of Electrical and Electronics Engineers, Computer*, 29(3):71–77, 1996.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Georg Thimm and Emile Fiesler. Pruning of neural networks. Technical report, Idiap Research Institute, 1997.

Hans H. Thodberg. Improving generalization of neural networks through pruning. *International Journal of Neural Systems*, 1(4):317–326, 1991.

Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. *Advances in Neural Information Processing Systems*, 7:505–512, 1995.

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

Hiroshi Tsukimoto. Extracting rules from trained neural networks. *Institute of Electrical and Electronics Engineers Transactions on Neural Networks and Learning Systems*, 11(2):377–389, 2000.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision, ECCV 2014, 13th European Conference, Zurich, Switzerland, Proceedings, Part 1*, pages 818–833, 2014.

Zhi-Hua Zhou, Shi-Fu Chen, and Zhao-Qian Chen. A statistics based approach for extracting priority rules from trained neural networks. In *International Joint Conference on Neural Networks, Proceedings*, volume 3, pages 401–406. Institute of Electrical and Electronics Engineers, 2000.

Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. Deepred - rule extraction from deep neural networks. In *Discovery Science - 19th International Conference, DS 2016, Bari, Italy, Proceedings*, pages 457–473, 2016.

A Annex

A.1 Predictive Accuracy Results when Using a Subset of the Data

Dataset	NR ND	NR LTP	WSP ND	WSP LTP	PP ND	PP LTP	PP MC	C4.5
$F1$	0.8978_4	0.9652_2	0.9957_1	0.9348_3	0.9957_1	0.9957_1	0.9957_1	0.9957_1
$F2$	0.8947_5	0.8947_5	0.9805_2	0.9815_1	0.9815_1	0.9815_1	0.9685_3	0.9663_4
$F3$	0.5614_2	0.5614_2	0.5614_2	0.8421_1	0.5614_2	0.8421_1	0.8421_1	0.5614_2
$F4$	0.8578_6	0.9121_4	0.9522_2	0.8893_5	0.9935_1	0.9935_1	0.9935_1	0.9327_3
$F5$	0.7913_1	0.7217_2	0.7913_1	0.7217_2	0.7913_1	0.7217_2	0.7913_1	0.7913_1
$F6$	0.5172_3	0.7931_1	0.5000_4	0.7931_1	0.5345_2	0.5345_2	0.4828_5	0.4828_5
$F7$	0.9897_1	0.9897_1	0.9897_1	0.9897_1	0.9897_1	0.9897_1	0.9897_1	0.9897_1
$F8$	0.9652_1	0.9652_1	0.9652_1	0.9652_1	0.9652_1	0.9652_1	0.9652_1	0.9652_1
$F9$	0.9839_3	0.9644_5	0.9935_1	0.9935_1	0.9876_2	0.9876_2	0.9876_2	0.9710_4
$F10$	0.7304_2	0.8522_1	0.7304_2	0.7304_2	0.7304_2	0.8522_1	0.6957_3	0.7304_2
$F11$	0.9697_1	0.9697_1	0.8355_4	0.8355_4	0.9697_1	0.8398_3	0.9697_1	0.8442_2
$F12$	0.9130_1	0.9130_1	0.9130_1	0.9130_1	0.9130_1	0.9130_1	0.8348_3	0.8522_2
$F13$	0.7652_1	0.7565_2	0.7565_2	0.7565_2	0.7304_3	0.7304_3	0.7565_2	0.7565_2
$F14$	0.7652_3	0.7652_3	0.9739_1	0.9739_1	0.7565_4	0.7565_4	0.8870_2	0.7565_4
$F15$	0.4912_2	0.4912_2	0.7018_1	0.7018_1	0.7018_1	0.7018_1	0.4912_2	0.4912_2
$F16$	0.9087_4	0.9087_4	0.9630_1	0.9630_1	0.9630_1	0.9630_1	0.9217_3	0.9457_2
$F17$	0.8804_5	0.8804_5	0.9435_3	0.9435_3	0.9913_1	0.9739_2	0.9435_3	0.9283_4
$F18$	0.9217_1	0.7043_3	0.9217_1	0.9217_1	0.9217_1	0.9217_1	0.9217_1	0.7913_2
$F19$	0.9951_1	0.9951_1	0.9951_1	0.9951_1	0.9951_1	0.9951_1	N/A	0.9951_1
$F20$	0.4609_3	0.4565_4	0.4565_4	0.4609_3	0.4609_3	0.4565_4	0.4739_1	0.4670_2

Table A.1.: Predictive accuracy when using 10% of the data, per dataset. Post-pruning was applied when building all models. The original scores were rounded to four decimal units. The subscript numbers order the approaches in terms of decreasing performance. The abbreviations in the model names indicate the following: *NR* stands for no retraining, *WSP* for weight sparseness pruning, *PP* for polarisation and weight sparseness pruning, *ND* for no offline discretisation, *LTP* for layerwise threshold preselection and *MC*, for activation range centre.

Dataset	NR ND	NR LTP	WSP ND	WSP LTP	PP ND	PP LTP	PP MC	C4.5
$F1$	0.9922_1	0.9922_1	0.9922_1	0.9245_2	0.9922_1	0.9922_1	0.9922_1	0.9922_1
$F2$	0.9635_6	0.9818_1	0.9805_2	0.9792_3	0.9792_3	0.9792_3	0.9661_4	0.9648_5
$F3$	0.9792_1	0.9792_1	0.9792_1	0.9792_1	0.9792_1	0.8542_2	0.9792_1	0.6667_3
$F4$	0.9284_4	0.8932_5	0.9948_1	0.9531_3	0.9948_1	0.9948_1	0.9948_1	0.9898_2
$F5$	0.8542_3	0.9063_1	0.8958_2	0.7500_4	0.8958_2	0.8958_2	0.8958_2	0.8542_3
$F6$	0.7708_3	0.7708_3	0.8958_2	0.8958_2	0.9792_1	0.9792_1	0.9792_1	0.8958_2
$F7$	0.9915_2	0.9915_2	0.9915_2	0.9915_2	0.9915_2	0.9915_2	0.9915_2	0.9935_1
$F8$	0.9688_2	0.9688_2	0.9896_1	0.9896_1	0.9896_1	0.9896_1	0.9635_3	0.9635_3
$F9$	0.9927_3	0.9970_1	0.9940_2	0.9940_2	0.9940_2	0.9883_4	0.9883_4	0.9826_5
$F10$	0.9271_1	0.9271_1	0.8542_3	0.8542_3	0.9167_2	0.9167_2	0.9167_2	0.8542_3
$F11$	0.8281_6	0.9427_3	0.8542_5	0.8542_5	0.9688_2	0.9688_2	0.9792_1	0.8958_4
$F12$	0.9167_2	0.8542_4	0.9167_2	0.9063_3	0.9167_2	0.9167_2	0.9271_1	0.8177_5
$F13$	0.8958_4	0.9271_3	0.9688_1	0.8646_5	0.9271_3	0.9271_3	0.9479_2	0.8958_4
$F14$	0.8646_2	0.8646_2	0.9479_1	0.9479_1	0.8646_2	0.8646_2	0.9479_1	0.8646_2
$F15$	0.7292_2	0.7292_2	0.9583_1	0.9583_1	0.7292_2	0.7292_2	0.7292_2	0.7292_2
$F16$	0.9557_2	0.9401_4	0.9557_2	0.9557_2	0.9349_5	0.9661_1	0.9453_3	0.9401_4
$F17$	0.8776_6	0.9193_4	0.9557_2	0.9010_5	0.9922_1	0.9922_1	0.9922_1	0.9375_3
$F18$	0.9167_3	0.9167_3	0.9479_2	0.9479_2	0.9063_4	0.9167_3	0.9688_1	0.9063_4
$F19$	0.9854_2	0.9824_3	0.9948_1	0.9948_1	0.9948_1	0.9948_1	N/A	0.9948_1
$F20$	0.3958_4	0.4010_3	0.4167_2	0.4375_1	0.3698_7	0.3802_6	0.3906_5	0.3906_5

Table A.2.: Predictive accuracy when using 25% of the data. The same clarifications and abbreviations hold as in Table A.1.

Dataset	NR ND	NR LTP	WSP ND	WSP LTP	PP ND	PP LTP	PP MC	C4.5
$F1$	0.9961_2	1.0000_1	1.0000_1	0.9414_3	1.0000_1	1.0000_1	1.0000_1	1.0000_1
$F2$	0.9785_1	0.9453_4	0.9727_3	0.9727_3	0.9727_3	0.9727_3	0.9766_2	0.9727_3
$F3$	0.9375_1	0.8750_2	0.9375_1	0.9375_1	0.9375_1	0.9375_1	0.8438_3	0.9375_1
$F4$	0.9609_4	0.9629_3	0.9941_1	0.9531_5	0.9941_1	0.9941_1	0.9941_1	0.9863_2
$F5$	0.9219_5	0.9219_5	0.9219_5	0.9531_3	0.9844_1	0.9375_4	0.9688_2	0.9531_3
$F6$	0.9375_1	0.9375_1	0.9375_1	0.8750_2	0.9375_1	0.9375_1	0.9375_1	0.9375_1
$F7$	0.9902_3	0.9902_3	0.9912_2	0.9912_2	0.9912_2	0.9912_2	0.9912_2	0.9961_1
$F8$	0.9844_1	0.9844_1	0.9766_2	0.9766_2	0.9766_2	0.9766_2	0.9609_3	0.9453_4
$F9$	0.9924_3	0.9963_1	0.9933_2	0.9933_2	0.9879_4	0.9641_6	0.9879_4	0.9817_5
$F10$	0.9531_1	0.9219_2	0.9531_1	0.9219_2	0.9063_3	0.9063_3	0.9219_2	0.9531_1
$F11$	0.9844_1	0.9844_1	0.9844_1	0.9063	0.9844_1	0.9844_1	0.9766_2	0.9453_3
$F12$	0.9141_5	0.8984_6	0.9609_2	0.9297_3	0.9844_1	0.9219_4	0.8984_6	0.8984_6
$F13$	0.9219_3	0.9219_3	0.9219_3	0.9219_3	0.9375_2	0.9688_1	0.9688_1	0.9375_2
$F14$	0.9844_1	0.9844_1	0.9688_2	0.9688_2	0.9844_1	0.9844_1	0.9844_1	0.9375_3
$F15$	0.9375_1	0.9375_1	0.9375_1	0.9375_1	0.9375_1	0.9375_1	0.9375_1	0.9375_1
$F16$	0.9766_2	0.9531_4	0.9961_1	0.9961_1	0.9961_1	0.9648_3	0.9531_4	0.9648_3
$F17$	0.9492_3	0.9492_3	0.9961_1	0.9258_4	0.9961_1	0.9961_1	0.9961_1	0.9531_2
$F18$	0.9063_3	0.9063_3	0.9219_2	0.9219_2	0.8906_4	0.8906_4	0.9688_1	0.8906_4
$F19$	0.9946_1	0.9946_1	0.9946_1	0.9946_1	0.9946_1	0.9946_1	N/A	0.9946_1
$F20$	0.3281_3	0.3203_4	0.3828_2	0.3984_1	0.3203_4	0.3047_6	0.3125_5	0.2891_7

Table A.3.: Predictive accuracy when using 50% of the data. The same clarifications and abbreviations hold as in Table A.1.