ORIGINAL RESEARCH
published: 22 October 2021
doi: 10.3389/frai.2021.689398

An Empirical Investigation Into Deep and Shallow Rule Learning

Florian Beck and Johannes Fürnkranz*

Institute for Application-oriented Knowledge Processing (FAW), Johannes Kepler University, Linz, Austria

Inductive rule learning is arguably among the most traditional paradigms in machine learning. Although we have seen considerable progress over the years in learning rule-based theories, all state-of-the-art learners still learn descriptions that directly relate the input features to the target concept. In the simplest case, concept learning, this is a disjunctive normal form (DNF) description of the positive class. While it is clear that this is sufficient from a logical point of view because every logical expression can be reduced to an equivalent DNF expression, it could nevertheless be the case that more structured representations, which form deep theories by forming intermediate concepts, could be easier to learn, in very much the same way as deep neural networks are able to outperform shallow networks, even though the latter are also universal function approximators. However, there are several non-trivial obstacles that need to be overcome before a sufficiently powerful deep rule learning algorithm could be developed and be compared to the state-of-the-art in inductive rule learning. In this paper, we therefore take a different approach: we empirically compare deep and shallow rule sets that have been optimized with a uniform general mini-batch based optimization algorithm. In our experiments on both artificial and real-world benchmark data, deep rule networks outperformed their shallow counterparts, which we take as an indication that it is worth-while to devote more efforts to learning deep rule structures from data.

Keywords: inductive rule learning, deep learning, learning in logic, mini-batch learning, stochastic optimization

OPEN ACCESS

Edited by:
Sophie Burkhardt,
Johannes Gutenberg University Mainz, Germany

Reviewed by:
Fabio Aiolli,
University of Padua, Italy
Joao Gama,
University of Porto, Portugal

****Correspondence:***
Florian Beck
fbeck@faw.jku.at

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: *31 March 2021*
Accepted: *10 September 2021*
Published: *22 October 2021*

Citation:
Beck F and Fürnkranz J (2021) An Empirical Investigation Into Deep and Shallow Rule Learning. Front. Artif. Intell. 4:689398. doi: 10.3389/frai.2021.689398

1 INTRODUCTION

Dating back to the AQ algorithm (Michalski, 1969), inductive rule learning is one of the most traditional fields in machine learning. However, when reflecting upon its long history (Fürnkranz et al., 2012), it can be argued that while modern methods are somewhat more scalable than traditional rule learning algorithms (see, e.g., Wang et al., 2017; Lakkaraju et al., 2016), no major break-through has been made. In fact, the Ripper rule learning algorithm (Cohen, 1995) is still very hard to beat in terms of both accuracy and simplicity of the learned rule sets. All these algorithms, traditional or modern, typically provide flat lists or sets of rules, which directly relate the input variables to the desired output. In concept learning, where the goal is to learn a set of rules that collectively describe the target concept, the learned set of rules can be considered as a logical expression in disjunctive normal form (DNF), in which each conjunction forms a rule that predicts the positive class.

In this paper, we argue that one of the key factors for the strength of deep learning algorithms is that latent variables are formed during the learning process. However, while neural networks excel in implementing this ability in their hidden layers, which can be effectively trained via backpropagation, there is essentially no counter-part to this ability in inductive rule learning. We therefore set out

A
```
parity :-     x1,     x2,     x3,     x4, not x5.
parity :-     x1,     x2, not x3, not x4, not x5.
parity :-     x1, not x2,     x3, not x4, not x5.
parity :-     x1, not x2, not x3,     x4, not x5.
parity :- not x1,     x2, not x3,     x4, not x5.
parity :- not x1,     x2,     x3, not x4, not x5.
parity :- not x1, not x2,     x3,     x4, not x5.
parity :- not x1, not x2, not x3, not x4, not x5.
parity :-     x1,     x2,     x3, not x4,     x5.
parity :-     x1,     x2, not x3,     x4,     x5.
parity :-     x1, not x2,     x3,     x4,     x5.
parity :- not x1,     x2,     x3,     x4,     x5.
parity :- not x1, not x2, not x3,     x4,     x5.
parity :- not x1, not x2,     x3, not x4,     x5.
parity :- not x1,     x2, not x3, not x4,     x5.
parity :-     x1, not x2, not x3, not x4,     x5.
```

B
```
parity45   :-     x4,     x5.
parity45   :- not x4, not x5.

parity345  :-     x3, not parity45.
parity345  :- not x3,     parity45.

parity2345 :-     x2, not parity345.
parity2345 :- not x2,     parity345.

parity     :-     x1, not parity2345.
parity     :- not x1,     parity2345.
```

FIGURE 1 | Unstructured and structured rule sets for the parity concept. **(A)** A flat unstructured rule set for the parity concept. **(B)** A deep unstructured rule base for parity using three auxiliary predicates.

to verify the hypothesis that deep rule structures might be easier to learn than flat rule sets, in very much the same way as deep neural networks have a better performance than single-layer networks (Mhaskar et al., 2017). Note that this is not obvious, because, in principle, every logical formula can be represented with a DNF expression, which corresponds to a flat rule set, in the same way as, in principle, one (sufficiently large) hidden layer is sufficient to approximate any function with a neural network (Hornik, 1991). As no direct comparison is possible because of the lack of a powerful algorithm for learning deep rule sets, our tool of choice is a simple stochastic optimization algorithm to optimize a rule network of a given size. While this does not quite reach state-of-the-art performance (in either setting, shallow or deep), it nevertheless allows us to gain some insights into these settings. In particular, we aim to see whether deep structures can be better learned than shallow structure in an identical setting using the same general optimization algorithm. To that end, we test deep and shallow rule networks on both, real-world UCI benchmark datasets, as well as artificial datasets for which we know the underlying target concept representations. Moreover, we also briefly look at the interpretability of the learned concepts in both their learned structure as well as their equivalent DNF formulation, but find that the presentation of logical formulas in a human interpretable way is still largely an open question.

The remainder of the paper is organized as follows: *Deep Rule Learning* elaborates why deep rule learning is of particular interest and refers to related work. We propose a new network approach in *Deep Rule Networks* and test it in *Experiments*. The results are concluded in *Conclusion*, followed by possible future extensions and improvements in *Future Work*.

2 DEEP RULE LEARNING

In this section, we will briefly discuss the state-of-the-art in learning deep, structured rule bases. We start with a brief motivation, and continue to review related work in several relevant areas, including constructive induction, multi-label rule learning, or binary and ternary networks.

2.1 Motivation

Rule learning algorithms typically provide flat lists that directly relate the input to the output. Consider, e.g., the following example: the parity concept, which is known to be hard to learn for heuristic, greedy learning algorithms, checks whether an odd or an even number of R relevant attributes (out of a possibly higher total number of attributes) are set to `true`. **Figure 1A** shows a flat rule-based representation[1] of the target concept for $R = 5$, which requires $2^{R-1} = 16$ rules. On the other hand, a structured representation, which introduces three auxiliary predicates (`parity2345`, `parity345` and `parity45` as shown in **Figure 1B**), is much more concise using only $2 \cdot (R - 1) = 8$ rules. We argue that the parsimonious structure of the latter could be easier to learn because it uses only a linear number of rules, and slowly builds up the complex target concept parity from the smaller subconcepts `parity2345`, `parity345` and `parity45`.

To motivate this, we draw an analogy to neural network learning, and view rule sets as networks. Conventional rule learning algorithms learn a flat rule set of the type shown in **Figure 1A**, which may be viewed as a concept description in disjunctive normal form (DNF): Each rule body corresponds to a single conjunct, and these conjuncts are connected via a disjunction (each positive example must be covered by one or more of these rule bodies). This situation is illustrated in **Figure 2A**, where the five input nodes are connected to 16 hidden nodes - one for each of the 16 rules that define the concept - and these are then connected to a single output node. Analogously, the deep parity rule set of **Figure 1B** may be encoded into a deeper network structure as shown in **Figure 2B**. Clearly, the

[1]We use a Prolog-like notation for rules, where the consequent (the head of the rule) is written on the left and the antecedent (the body) is written on the right. For example, the first rule reads as: If x1, x2, x3 and x4 are all true and x5 is false then parity holds

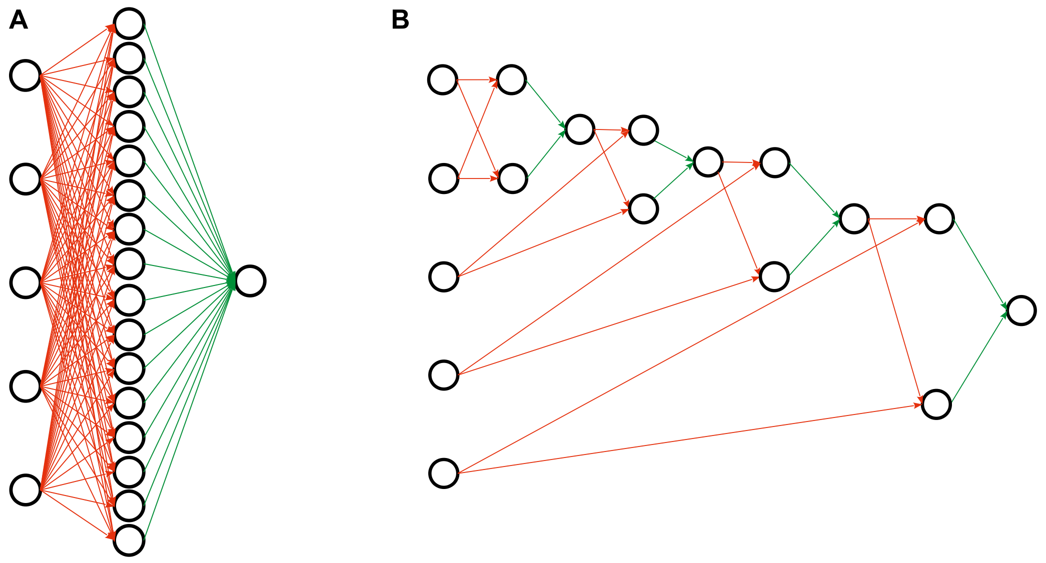

FIGURE 2 | Network representations of the parity rule sets of **Figure 1**. Red connections are logical ANDs, green edges correspond to logical ORs. **(A)** Shallow representation. **(B)** Deep representation.

deep network is more compact and considerably sparser in the number of edges. Of course, we need to take into consideration that the optimal structure is not known beforehand and presumably needs to emerge from a fixed network structure that offers the possibility for some redundancy, but nevertheless we expect that such structured representations offer similar advantages as deep neural networks offer over single-layer networks.

It is important to note that deep structures do not increase the expressiveness of the learned concepts. Any formula in propositional logic (and we limit ourselves to propositional logic in this project) can be converted to a DNF formula. In the worst case (a so-called *full DNF*), each of the input variables appears exactly once in all of the inputs, which essentially corresponds to enumerating all the positive examples. Thus, the size of the number of conjuncts in a DNF encoding of the inputs may grow exponentially with the number of input features. This is in many ways analogous to the universal approximation theorem (Hornik, 1991), which essentially states that any continuous function can be approximated arbitrarily closely with a shallow neural network with a single hidden layer, provided that the size of this layer is not bounded. So, in principle, deep neural networks are not necessary, and indeed, much of the neural network research in the 90s has concentrated on learning such two-layer networks. Nevertheless, we have now seen that deep neural networks are easier to train and often yield better performance, presumably because they require exponentially less parameters than shallow networks (Mhaskar et al., 2017). In the same way, we expect that deep logical structures will yield more efficient representations of the captured knowledge and might be easier to learn than flat DNF rule sets.

2.2 State-Of-The-Art in Deep Rule Learning

As mentioned above, the problem of deep rule learning has only rarely been explicitly addressed in the literature. Modern rule learning algorithms rely on ensemble-based sequential loss minimization. Friedman and Popescu (2008), e.g., learn a sparse linear model from features that have been obtained from the rules corresponding to the leaves of a decision tree ensemble such as a random forest (Breiman, 2001). Algorithms like ENDER (Dembczyński et al., 2010) or BOOMER (Rapp et al., 2020) integrate the rule induction into the boosting procedure by aiming at the minimization of an overall regularized loss function during the learning of individual rules. The learning algorithm for finding interpretable decision sets (Lakkaraju et al., 2016) explicitly includes several biases for interpretability into the objective function and proposes smooth stochastic search, a method for efficiently finding an approximative solution. Angelino et al. (2017) demonstrate an algorithm that is able to find exact loss minimizing rules.

All these algorithms are single-concept learners, i.e., they learn rules for a single target concept. However, as has been argued by Fürnkranz et al. (2020), works in several related areas are quite relevant to the problem. In the following, we briefly review approaches that are able to convert deep models into rules (*Rule Extraction From Deep Models*), to autonomously discover hidden, auxiliary concepts (*Learning Intermediate Concepts*), or to learn multiple dependent target concepts (*Learning Multiple Dependent Concepts*), and even review a few algorithms that learn logical networks (*Discrete Deep Networks*).

2.2.1 Rule Extraction From Deep Models

The strength of many recent learning algorithms, most notably deep learning (Lecun et al., 2015; Schmidhuber, 2015), but also kernel-based methods (Cristianini and Shawe-Taylor, 2000) is that the input variables are combined to form latent concepts during the learning process. Understanding the meaning of these hidden variables is crucial for transparent and justifiable decisions. Consequently, some research has been devoted to trying to convert arcane models such as neural networks (Andrews et al., 1995; Craven and Shavlik, 1997; Schmitz et al., 1999) or support-vector machines (Barakat and Bradley, 2010; Guerreiro and Trigueiros, 2010) to more interpretable rule-based or tree-based models. Nevertheless, these models typically treat the input models as black-boxes, and do not try to uncover the structure in the hidden layers. One exception is, e.g., DeepRED (Zilke et al., 2016 ; González et al., 2017), which tries to learn a decision tree model for each node in a neural network, but eventually compiles them into a single flat rule set. In a particularly interesting recent work, Polato and Aiolli (2019) defined kernels based on propositional logic, which allowed them to demonstrate several examples of the hardness of explicitly formulating some of the learned concepts (such as "three cards of a kind" in poker) in pure Boolean logic.

Instead of making the entire model interpretable, methods like lime (Ribeiro et al., 2016) are able to provide local explanations for inscrutable models, allowing to trade off fidelity to the original model with interpretability and complexity of the local model. For example, lore (Guidotti et al., 2018) explicitly targets rule-based explanations of black-box models. An interesting aspect of rule-based theories is that they can be considered as hybrids between local and global explanations (Fürnkranz, 2005): A rule set may be viewed as a global model, whereas the individual rule that fires for a particular example may be viewed as a local explanation. A recent method, glocalx (Setzu et al., 2021), conversely, aims to combine local explanations into a global rule model. However, again, these approaches are not able to capture the subconcepts detected by the black-box classifiers.

2.2.2 Learning Intermediate Concepts

For learning structured rule sets, a key challenge is how to define and train the intermediate, hidden concepts h_i which may be used for improving the final prediction. Note that in a conventional, flat structure as in **Figure 2A**, all h_i always had a fairly clear semantic in that they capture some aspect of the target variable y. The rule that predicts h_i essentially defines a local pattern for y (Fürnkranz, 2005).

However, when learning deeper structures, other hidden concepts need to be defined which do not directly correspond to the target variable, as can be seen from the structured parity concept of **Figure 1B**. This line of work has been known as *constructive induction* (Matheus, 1989) or *predicate invention* (Stahl, 1996), but surprisingly, it has not received much attention since the classical works in inductive logic programming in the 1980s and 1990s. One approach is to use a wrapper to scan for regularly co-occurring patterns in rules, and use them to define new intermediate concepts which allow to compress the original theory (Pfahringer, 1994; Wnek and Michalski, 1994). Alternatively, one can directly invoke so-called predicate invention operators during the learning process, as, e.g., in Duce (Muggleton, 1987), which operates in propositional logic, and its successor systems in first-order logic (Muggleton and Buntine, 1988; Kijsirikul et al., 1992; Kok and Domingos, 2007). Similar to Duce, systems like MOBAL (Morik et al., 1993) not only try to learn theories from data, but also provide functionalities for reformulating and restructuring the knowledge base (Sommer, 1996). More recently, Muggleton et al. (2015) introduce a technique that employs user-provided meta rules for proposing new predicates, which allow it to invent useful predicates from only very training examples. Kramer (2020) provides an excellent recent summary of work in this area.

2.2.3 Learning Multiple Dependent Concepts

Much of the work in machine learning is devoted to single prediction tasks, i.e., to tasks where an input vector is mapped to a single output value. When aiming to learn a deep rule base, however, one has to tackle the problem of learning a network of multiple, possibly mutually dependent concepts. A pioneering work in this area is Malerba et al. (1997), which gives a broad discussion of the problem of learning multiple dependent concepts in the form of a dependency graph. Back then, the problem has primarily been studied in inductive logic programming and relational learning (see, e.g., De Raedt et al., 1993), but it has recently reappeared in multilabel classification (Tsoumakas and Katakis, 2007; Tsoumakas et al., 2010; Zhang and Zhou, 2014) and, more generally, in multi-target prediction (Waegeman et al., 2019).

In fact, most of the research in multi-label classification aims for the development of methods that are capable of modeling label dependencies (Dembczyński et al., 2012). One of the best-known approaches are so-called classifier chains (CC) (Read et al., 2011, 2021), which learn the labels in some (arbitrary) order where the predictions for previous labels are included as features for subsequent models. Several extensions of this framework have been studied, such as Burkhardt and Kramer (2015), who propose to cluster labels into sequential blocks of sets of labels. A general framework proposed by Read and Hollmén (2014, 2015) formulates multi-label classification problems as deep networks where label nodes are a special type of hidden nodes which can appear in multiple layers of the networks.

However, while these algorithms all aim at learning multiple interconnected models, they are not capable of explicitly defining intermediate, auxiliary concepts. Some works that aim at finding so-called label embeddings (e.g., Nam et al., 2016) may be viewed in this context, but they do not learn rule-based descriptions. Rules are particularly interesting for solving this kind of problems because they allow to explicitly formalize and model dependencies between labels and between data and labels in an explicit and seamless way (Hüllermeier et al., 2020). Rapp et al. (2020) propose an efficient boosting-based rule learner for multi-label classification.

2.2.4 Discrete Deep Networks

Finally, in the wake of the success of deep neural networks, a few approaches have been developed that explicitly aim at learning networks with a logical structure. Sum-product networks (SPNs; Poon and Domingos, 2011) have an analogous structure to our AND/OR networks, but aim at modeling probability distributions instead of logical expressions. Of particular interest to our study is the work of Delalleau and Bengio (2011), who compare deep and shallow SPNs, and find that deep structures can result in more compact representations, which is in line with the motivation of our work.

Somewhat closer to logic are frameworks such as TensorLog (Cohen et al., 2020) that aim at making probabilistic logical reasoning differentiable and therefore amenable to implementation and optimization in a deep learning environment. For example, the approach of Evans and Grefenstette (2018) is able to learn logical theories from data in a matter that is considerably more robust than traditional techniques from inductive logic programming. However, it only learns to weight rules that can be generated from a set of predefined templates. In particular, no auxiliary, hidden predicates can emerge from the learner. Fuzzy pattern trees (Senge and Hüllermeier, 2011) may be viewed in this way in that they build up a hierarchical structure of generalized logical functions, so that their internal nodes may be viewed as intermediate fuzzy logical concepts.

Most relevant to our work are binary networks (Courbariaux et al., 2015; Qin et al., 2020), which restrict the weights to values $\{-1, 1\}$. However, their semantics does typically not correspond to conventional logic rules, in that they enforce every feature to contribute to the function to be learned, either in its positive or negated form. Ternary networks (Li et al., 2016; Zhu et al., 2017), with weights $\{-1, 0, 1\}$, where 0 corresponds to ignoring the corresponding feature in the rule, could provide a solution to this, and are, indeed, quite similar in spirit to the networks we train in the remainder of this paper. Typically, they train a full deep neural network, and subsequently quantize the resulting weights to the desired two or three values, in order to allow a more compact representation and faster inference. Nevertheless, we have not made use of them in our work, because we wanted to focus on a simple optimization algorithm that is invariant for deep and shallow structures. For essentially the same reason, we have also not used state-of-the-art flat rule learning algorithms, so that observed differences in performance can be clearly attributed to differences in the network structure, and not in the optimization algorithms.

3 DEEP RULE NETWORKS

For our studies of deep and shallow rule learning, we define rule-based theories in a networked structure, which we describe in the following. We build upon the shallow two-level networks which we have previously used for experimenting with mini-batch rule learning (Beck and Fürnkranz, 2020), but generalize them from a shallow DNF-structure to deeper networks.

3.1 Network Structure

A conventional rule set consisting of multiple conjunctive rules that define a single target concept, corresponds to a logical expression in disjunctive normal form (DNF). An equivalent network consists of three layers, the input layer, one hidden layer (= AND layer) and the output layer (= OR layer), as, e.g., illustrated in **Figure 2A**. The input layer receives one-hot-encoded nominal attribute-value pairs as binary features (= literals), the hidden layer conjuncts these literals to rules and the output layer disjuncts the rules to a rule set. The network is designed for binary classification problems and produces a single prediction output that is `true` if and only if an input sample is covered by any of the rules in the rule set.

For generalizing this structure to deeper networks, we need to define multiple layers. While the input layer and the output layer remain the same, the number and the size of the hidden layers can be chosen arbitrarily. In the more general case, the hidden layers are treated alternately as conjunctive and disjunctive layers. We focus on networks with an odd number of hidden layers, starting with a conjunctive hidden layer and ending with a disjunctive output layer. In this way, the output will be easier to compare with rule sets in DNF. Furthermore, the closer we are to the output layer, the more extensive are the rules and rule sets, and the smaller is the chance to form new combinations from them that are neither tautological nor contradictory. As a consequence, the number of nodes per hidden layer should be lower the closer it is to the output layer. This makes the network shaped like a funnel.

3.2 Network Weights and Initialization

In the following, we assume the network to have $n + 2$ layers, with each layer i containing s_i nodes. Layer 0 corresponds to the input layer with $s_0 = |\mathrm{x}|$ and layer $n + 1$ to the output layer with $s_{n+1} = 1$. Furthermore, a weight $w_{jk}^{(i)}$ is identified by the layer i it belongs to, the node j from which it receives the output, and the node k in the successive layer $i + 1$ to which it passes the activation. Thus, the weights of each layer can be represented by an $s_i \times s_{i-1}$-dimensional matrix $W^{(i)} = [w_{jk}^{(i)}]$. In total, there are $\sum_{i=0}^{n} s_i s_{i+1}$ Boolean weights which have to be learned, i.e., have to be set to true (resp. 1) or false (resp. 0). If weight $w_{jk}^{(i)}$ is set to `true`, this means that the output of node j is used in the conjunction (if i mod 2 = 0) or disjunction (if i mod 2 = 1) that defines node k. If it is set to false, this output is ignored by node k.

In the beginning, these weights need to be initialized. This initialization process is influenced by two hyperparameters: average rule length ($\bar{l}$) and initialization probability (p), where $\bar{l}$ only affects the number of weights that are set to one in the first layer. Here we use the additional information which literals belong to the same attribute to avoid immediate contradictions within the first conjunction. Let $|\mathcal{A}|$ be the number of attributes, then each attribute is selected with the probability $\bar{l}/|\mathcal{A}|$ so that on average for $\bar{l}$ literals of different attributes the corresponding weight will be set to `true`. In the remaining layers, the weights are set to true with the probability p. Additionally, at least one outgoing weight from each node will be set to `true` to ensure connectivity. This implies that, regardless of the choice of p, all the weights in the last layer will always be initialized with `true` because there is only one output node. Note that, as a consequence, shallow DNF-structured networks will not be influenced by the choice of p, since they only consist of the first layer influenced by $\bar{l}$ and the last layer initialized with `true`.

3.3 Prediction

The prediction of the network can be efficiently computed using binary matrix multiplications ($\circledcirc$). In each layer i, the input features $A^{(i)}$ are multiplied with the corresponding weights

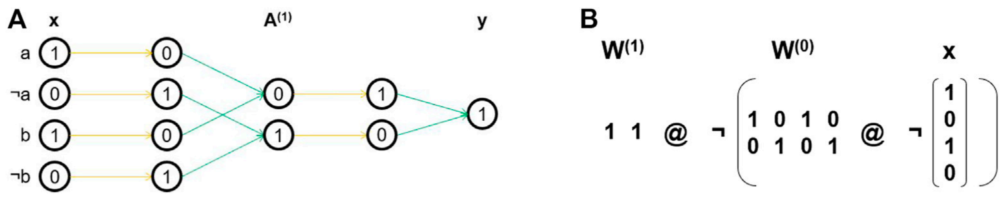

FIGURE 3 | Example network and binary matrix multiplications for the parity concept with two variables, $y = (a \wedge b) \vee (\neg a \wedge \neg b)$. Orange connections are logical negations (¬), green edges correspond to binary matrix multiplications (@). **(A)** Activations processed through the network. **(B)** matrix multiplication.

$W^{(i)}$ and aggregated at the receiving node in layer $i + 1$. If the aggregation is disjunctive, this directly corresponds to a binary matrix multiplication. Each product of an input feature $a^{(i)}$ and its corresponding weight $w^{(i)}$ is either `true` or false, and the summation of these products is set to `true` for any sum larger than zero, i.e., if any rule fires. Furthermore, according to De Morgan's law, $a \wedge b = \neg(\neg a \vee \neg b)$ holds. This means that binary matrix multiplication can be used also in the conjunctive case, provided that the inputs and outputs are negated before and after the multiplication. Because of the alternating sequence of conjunctive and disjunctive layers, binary matrix multiplications and negations are also always alternated when passing data through the network, so that a binary matrix multiplication followed by a negation can be considered as a NOR-node. Thus, the activations $A^{(i+1)}$ can be computed from the activations in the previous layers as

$$A^{(i+1)} \leftarrow \tilde{A}^{(i)} \odot W^{(i)} \tag{1}$$

where $\tilde{X} = J - X$ denotes the element-wise negation of a matrix X (J denotes a matrix of all ones). Hence, internally, we do not distinguish between conjunctive and disjunctive layers within the network, but have a uniform network structure consisting only of NOR-nodes. However, for the sake of the ease of interpretation, we chose to represent the networks as alternating AND and OR layers.

In the first layer, we have the choice whether to start with a disjunctive layer or a conjunctive one, which can be controlled by simply using the original input vector ($A^{(0)} = \mathrm{x}$) or its negation ($A^{(0)} = \tilde{\mathbf{x}}$) as the first layer. Also, if the last layer is conjunctive, an additional negation must be performed at the end of the network so that the output has the same "polarity" as the target values. In our experiments, we always start with a conjunctive and end with a disjunctive layer. In this way, the rule networks can be directly converted into conjunctive rule sets.

Figure 3 illustrates an example prediction for the parity concept with two variables, a and b, and two rules $y = a \wedge b$ and $y = \neg a \wedge \neg b$. Negations are marked by ¬ and orange arrows, Boolean matrix multiplications by @ and green arrows. In **Figure 3A**, the computed negations and activations for an input $x = \{a, b\}$ are stated within the nodes. The corresponding weight matrices and the complete prediction formula are shown in **Figure 3B**.

3.4 Training

Following Beck and Fürnkranz (2020), we implement a straight-forward mini-batch based greedy optimization scheme. While the number, the arrangement and the aggregation types of the nodes remain unchanged, the training process will flip the weights of the network to optimize its outcome. Flipping a weight from 0 to 1 (or vice versa) can be understood to be a single addition (or removal) of a literal to the conjunction or disjunction encoded by the following node.

Algorithm 1. Deep Rule Network Training, fit()-method

```
Input: drnc, $x$, $y$, batch_size
Output: drnc
drnc.initialize();
n_samples ← length($x$);
n_batches ← n_samples / batch_size;
best_accuracy ← accuracy_score($y$,
  drnc.predict($x$));
best_coefs ← drnc.coefs;
for $e$ ← 0 to n_epochs −1 do
    $x$, $y$ ← shuffle($x$, $y$);
    for $b$ ← 0 to n_batches −1 do
        $x_{mb}$, $y_{mb}$ ← next_batch(batch_size);
        coefs ← drnc.optimize_coefs($x_{mb}$, $y_{mb}$);
        accuracy ← accuracy_score($y$,
          drnc.predict($x$));
        if accuracy > best_accuracy then
            best_accuracy ← accuracy;
            best_coefs ← drnc.coefs;
        end
    end
end
drnc.coefs ← best_coefs;
drnc.optimize_coefs($x$, $y$);
return drnc
```

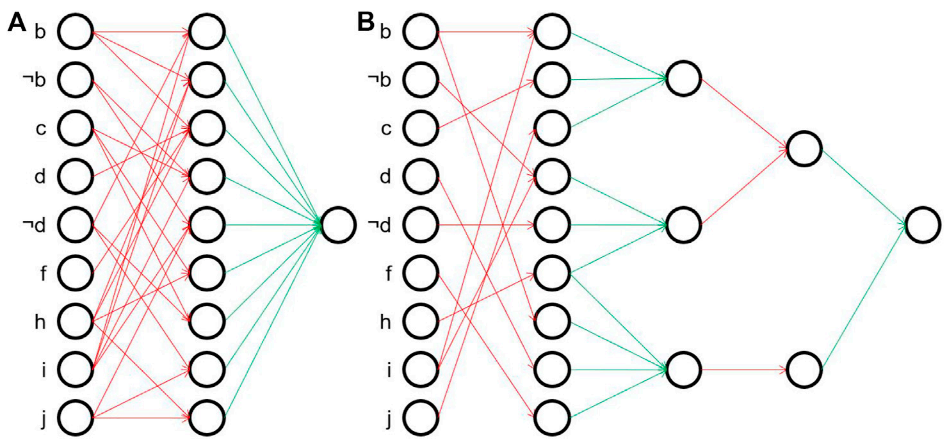

FIGURE 4 | Example networks of **Eq. 2**, **3**. For a better overview, the input nodes not influencing the final concept are removed. Red connections are logical ANDs, green edges correspond to logical ORs. **(A)** Shallow example. **(B)** Deep example.

Algorithm 2. Deep Rule Network Training, optimize_coefs()-method

```
Input: drnc, $x$, $y$, best_accuracy
Output: coefs
best_$i$, best_$j$, best_$k$ ← $i$, $j$, $k$;
optimal ← false;
flip_count ← 0;
while ¬ optimal ∧ flip_count < drnc.max_flips do
    optimal ← true;
    for $i$ ← 0 to n_layers −1 do
        for $j$ ← 0 to n_nodes[$i$] −1 do
            for $k$ ← 0 to n_nodes[$i-1$] −1 do
                flip(drnc.coefs[$i$][$j$][$k$]);
                accuracy ← accuracy_score($y$,
                  drnc.predict($x$));
                if accuracy > best_accuracy then
                    optimal ← false;
                    best_accuracy ← accuracy;
                    best_coefs ← drnc.coefs;
                    best_$i$, best_$j$, best_$k$ ← $i$, $j$, $k$;
                end
                flip(drnc.coefs[$i$][$j$][$k$]);
            end
        end
    end
    if ¬ optimal then
        flip(drnc.coefs[$i$][$j$][$k$]);
        best_accuracy ← accuracy;
        best_coefs ← drnc.coefs;
        flip_count ← flip_count +1;
    end
end
return best_coefs
```

A detailed pseudo-code of the training process is shown in **Algorithms 1** and **2**. Given a deep rule network classifier *drnc*, training samples *x*, their correct targets *y* and an appropriate batch size for the training set, **Algorithm 1** shows a naïve greedy approach to fit the network to the training data. After the initialization, the base accuracy on the complete training set and the initial weights are stored and subsequently updated every time when the predicted accuracy on the training set exceeds the previous maximum after processing a mini-batch of training examples. However, the predictive performance does not necessarily increase monotonically, since the accuracy is optimized not on the whole training set, but on a mini-batch. For all layers and nodes, possible flips are tried and evaluated, and the flip with the biggest improvement of the accuracy on the current mini-batch is selected. These greedy adjustments are repeated until either no flip improves the accuracy on the mini-batch or a maximum number of flips is reached, which ensures that the network does not overfit the mini-batch data.

When all mini-batches are processed, the procedure is repeated for a fixed number of epochs. Only the composition of the mini-batches is changed in each epoch by shuffling the training data before proceeding. After all epochs, the weights of the networks are reset to the optimum found so far, and a final optimization on the complete training set is conducted to eliminate any overfitting on mini-batches. The returned network can then be used to predict outcomes of any further test instances.

4 EXPERIMENTS

In this section we present the results of differently structured rule networks on both artificial and real-world UCI datasets, with the goal of investigating the effect of differences in the depth of the networks. We first describe the artificial datasets (*Artificial Datasets*), then some preliminary experiments that helped us

TABLE 1 | Hyperparameters for deep and shallow networks.

	Deep	shallow
n, s_i	5: (72, 36, 12, 6, 2), (32, 16, 8, 4, 2) 4: (36, 12, 6, 2), (16, 8, 4, 2) 3: (12, 6, 2), (8, 4, 2)	1: 10, 20, 50, 100, 200, 500
l	1, 2, 3	1, 2, 3, 4, 5, 6, 7
P	0.025, 0.075, 0.125	-

to focus on suitable network structures and hyperparameters (*Hyperparameter Tuning*), and finally discuss the main results on the artificial and real datasets. The code and the datasets are available in a public repository[2].

4.1 Artificial Datasets

As many standard UCI databases can be solved with very simple rules (Holte, 1993), we generated artificial datasets with a deep structure that we know can be represented by our network. An artificial dataset suitable for our greedy optimization algorithm should not only include intermediate concepts which are meaningful but also a strictly monotonically decreasing entropy between these concepts, so that they can be learned in a stepwise fashion in successive layers. One way to generate artificial datasets that satisfy these requirements is to take the output of a randomly generated deep rule network. Subsequently, this training information can be used to see whether the function encoded in the original network can be recovered. Note that such a recovery is also possible for networks with different layer structures. In particular, each of the logical functions encoded in such a deep network can, of course, also be encoded as a DNF expression, so that shallow networks are not in an a priori disadvantage (provided that their hidden layer is large enough, which we ensure in preliminary experiments reported in *Hyperparameter Tuning*).

We use a dataset of ten Boolean inputs named a to j and generate all possible 2^{10} combinations as training or test samples. These samples are extended by the ten negations $\neg a$ to $\neg j$ via one-hot-encoding and finally passed to a funnel-shaped deep rule network with $n = 5$ and s = [32, 16, 8, 4, 2]. The weights of the network are set by randomly initializing the network and then training it on two randomly selected examples, one assigned to the positive and one to the negative class, to ensure both a positive and negative output is possible. If the resulting ratio of positively predicted samples is still less than 20% or more than 80%, the network is reinitialized with a new random seed to avoid extremely imbalanced datasets.

An example concept for a generated dataset is shown in **Figure 4**. Thinking of a rule network, circles represent nodes and are connected by an arrow if and only if the corresponding weight is `true`. Note that many nodes and weights are irrelevant, and are not shown, e.g., neither a nor $\neg a$ have an influence on the generated output. The flat representation shown in **Figure 4A** corresponds to the following DNF expression[3]:

$$(b\wedge\neg d\wedge i)\vee(b\wedge h\wedge i)\vee(b\wedge d\wedge f\wedge h)\vee(\neg b\wedge c\wedge i)\vee(\neg b\wedge i\wedge j)\vee$$
$$(c\wedge h)\vee(c\wedge\neg d)\vee(\neg d\wedge j)\vee(h\wedge j) \qquad (2)$$

It can be clearly seen that the rules in this simplified formula share some common features, indicating intermediate concepts in subsequent layers which are combined in the end. A more compact representation of the same concept using a hierarchical structure is:

$$(((b\wedge i)\vee c\vee j)\wedge((\neg b\wedge i)\vee\neg d\vee h))\vee(b\wedge d\wedge f\wedge h) \qquad (3)$$

The second representation only needs 11 aggregations (6 AND, 5 OR) in comparison to 23 aggregations (15 AND, 8 OR) in the first one. This is also reflected in the number of weights set to `true`, i.e. the number of arrows in **Figure 4** (33 in **Figure 4A** vs. 26 in **Figure 4B**). In contrast, when training the deep rule network, we must learn at least 180 + 27 + 6 + 2 = 215 binary weights correctly, while for the shallow one already 180 + 9 = 189 would be sufficient. Note that we included here the eleven nodes in the input layer which are absent in the formulas, but whose weights nevertheless have to be learned by both networks. Looking at the figures, it is also noticeable that even though both networks have sparse weight matrices, the ones of the hierarchical network are even more sparse which makes it almost shaped like a tree. In the following experiments, we evaluate which of these two representations is easier to learn approximately.

4.2 Hyperparameter Tuning

Before the main experiments, we conducted a few preliminary experiments on three of the artificial datasets to set suitable default values for the hyperparameters of the networks. One of these hyperparameters also known from neural networks is the number of epochs (*n_epochs*). By definition (**Algorithm 1**), the accuracy monotonically increases with a higher number of epochs while at the same time the training time rises as well. After five epochs, the performance no longer rises remarkably, so this value seems as a good trade-off between performance and training time.

The second hyperparameter *batch_size* affects these two measures as well. After tests using the number of instances as *batch_size* and others skipping the final optimization in **Algorithm 1** on the full batch, we notice that using a combination of mini-batches and full batches performs better than either of the two individual batch variants. For the artificial datasets, a *batch_size* of 50 is suitable. Finally, the limitation of iterations per mini-batch by *max_flips* also influences both the accuracy and training time. However, in case of noise-free artificial data, we can leave *max_flips* unbounded to achieve the optimal performance.

The experiments also showed no clear advantages or disadvantages between a conjunctive or disjunctive first layer, so in the following experiments we focus on networks starting with a conjunctive layer which offer the biggest similarity and best comparability to models learned from classic rule learners. Furthermore, we dispense with a separate optimization of the last layer like in (Beck and Fürnkranz, 2020), as this did not result in any improvement in performance.

[2]https://github.com/f-beck/scikit-learn-rule-network

[3]We used the python library SYMPY (https://www.sympy.org/en/index.html) to compute the corresponding DNF representation

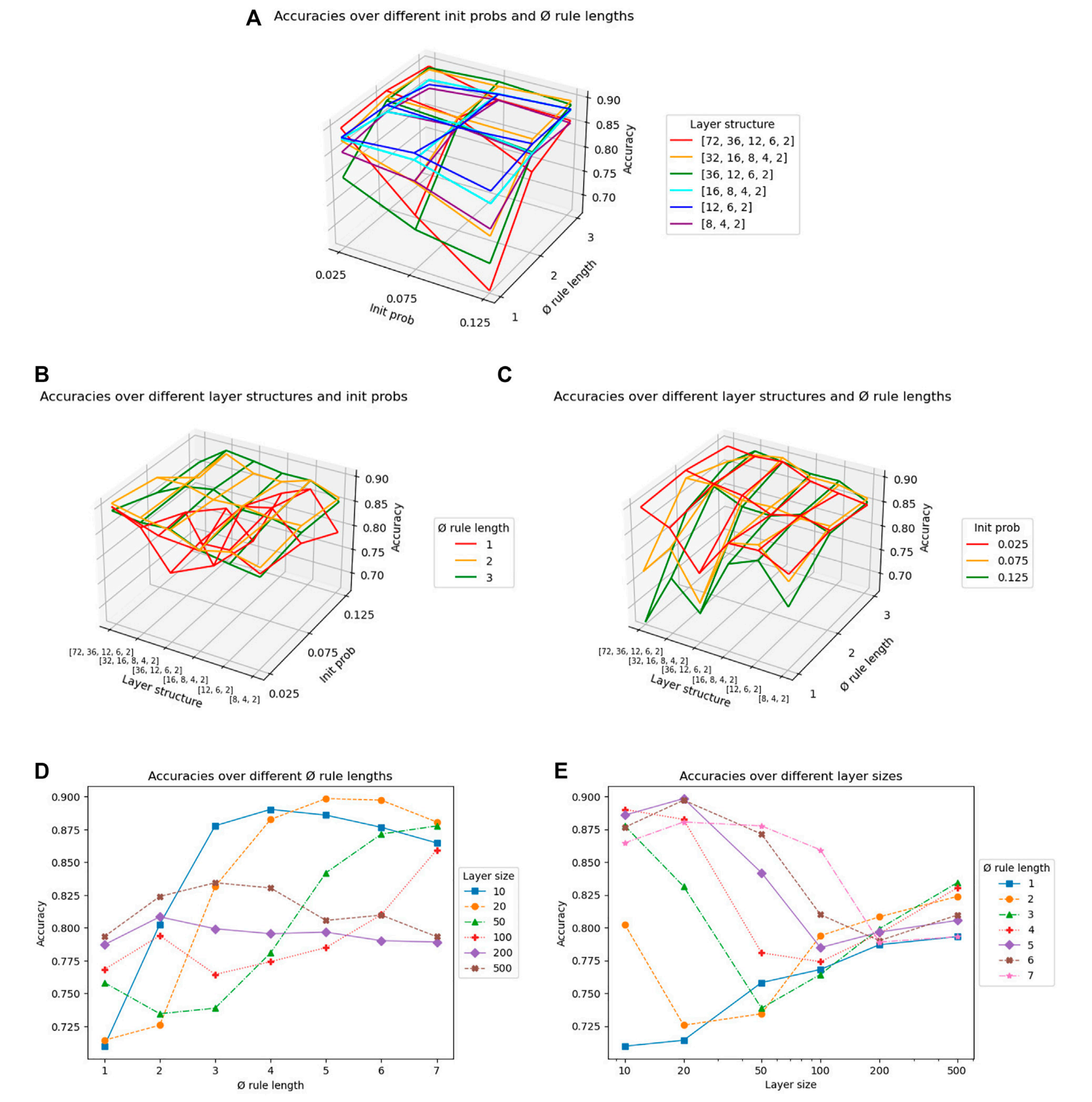

FIGURE 5 | Grid search on the hyperparameters average rule length $\bar{l}$, initialization probability p, number n and sizes s_i of hidden layers. The first three plots show accuracies for deep rule networks with different combinations of **(A)** p and $\bar{l}$, **(B)** $n|s_i$ and $\bar{l}$ and **(C)** $n|s_i$ and p. The last two show accuracies for shallow rule networks with different **(D)** $\bar{l}$ and **(E)** s_1.

For the remaining hyperparameters, we tried to find appropriate settings by performing a grid search on 20 artificial datasets. The hyperparameters to be optimized are the average rule length $\bar{l}$, the initialization probability p, the number n and the sizes s_i of hidden layers. The other hyperparameters are set to the default values stated above, except that only a single epoch is used in order to speed up the grid search. This will have a negative effect on the performance in general, but should not significantly change the ranking of the different networks.

Table 1 shows the hyperparameters that we compared in a grid search. For the deep networks, we set n to values from 3 to 5. On the one hand, this guarantees that they contain at least two conjunctive and two disjunctive layers to map a wide variety of hierarchical concepts effectively. On the other hand, it still permits that the values of s_i can be set to values bigger than 10 while maintaining a reasonable training time with the naïve greedy algorithm. We create two different networks for each of the three values of n and set the values s_i so that $s_i \geq 2s_{i+1}$, resulting in a smaller network and a bigger one containing 1.5 to 3.5 times as many weights that can be adapted. For shallow networks, n is by definition set to 1. To ensure that these networks have approximately the same expressive power as the corresponding deep networks, we set s_1 so that the total number of weights in both network types is roughly the same. Additionally, we try a very high number of $s_1 = 500$ rules to estimate if a very big single layer can improve the accuracy remarkably. For the average initialized rule length $\bar{l}$ we will use the integer values from one to three for deep networks and from one to seven for shallow ones. We assume that in shallow networks a higher value of $\bar{l}$ is required, while in deep networks the intermediate concepts can be combined in successive layers. The numerical deficit of deep network test cases caused by $\bar{l}$ is compensated by the additional hyperparameter p, where we use three values between O.O25 and O.125. Therefore, in total, the accuracy of the deep network will be mapped to three dimensions $n|s_i$, $\bar{l}$ and p and the accuracy of the shallow network to only two dimensions s_1 and $\bar{l}$.

The results of the grid search are shown in **Figure 5**. The optimal hyperparameter setting for deep networks with an accuracy of 0.9073 is s = (72, 36, 12, 6, 2), $\bar{l} = 2$ and $p = 0.025$. However, we notice in **Figure 5A** that the red curve of the largest structure s = (72, 36, 12, 6, 2) not only contains the maximum, but also the minimum accuracy with $\bar{l} = 1$ and $p = 0.125$. While in general the combination of a lower $\bar{l}$ and a higher p decreases the accuracy, this effect seems to be stronger the bigger the network structure is. Despite the higher sensitivity, the larger layer structures provide better maximum accuracies than the smaller ones, as can be seen in the upper left corner of the graph. When comparing the graphs for different values of $\bar{l}$ in **Figure 5B**, it is noticeable that, with only few exceptions, the red curve for $\bar{l} = 1$ lies below the other two curves. The same can be observed in **Figure 5C**, here for the green curve for $p = 0.125$. Combinations of other values of $\bar{l}$ and p provide good accuracies regardless of the layer structure.

For shallow networks, we can see a high correlation between s_1 and $\bar{l}$ in **Figure 5D**. The graphs (except of the red one) resemble a downward-opening parabola, so that the accuracy becomes lower and lower the greater the deviation from this optimal value. Thereby applies that the smaller the layer size, the larger is the optimal value of $\bar{l}$, e.g., 5 for 20 and 2 for 200. Finally, **Figure 5E** shows that, contrary to the deep networks, the sensitivity to $\bar{l}$ decreases with the size of the shallow network. The optimal accuracy of 0.8984 is reached when the shallow network hyperparameters are set to $s_1 = 20$ and $\bar{l} = 5$.

Based on the above results, we will only use three network versions for the main experiments reported in the following sections. As a candidate for shallow networks, we take the best combination of $s_1 = 20$ and $\bar{l} = 5$. For the deep networks, however, we will choose the second-best network s = (32, 16, 8, 4, 2) combined with $\bar{l} = 2$ and an averaged $p = 0.05$, since it is almost ten times faster than the best deep network while still reaching an accuracy over 0.895. The third network is chosen as an intermediate stage between the first two: s = (32, 8, 2) combined with $\bar{l} = 3$ and $p = 0.05$. While still being a deep network, the learned rules can be passed to the output layer a little faster. In the following, we will refer to these (deep) rule network classifiers based on their number of layers, i.e. DRNC(5) for s = (32, 16, 8, 4, 2), DRNC(3) for s = (32, 8, 2) and RNC for $s_1 = 20$. For computational reasons, all of the reported results were estimated with a 2-fold cross validation. While this may not yield the most reliable estimate on each individual dataset, we nevertheless get a coherent picture over all 20 datasets, as we will see in the following.

4.3 Results on Artificial Datasets

In the main experiments, we use a combination of 15 artificial datasets with seeds we already used in a prior hyperparameter grid search and five artificial datasets with new seeds to detect potential overfitting on the first datasets. All datasets are tested using five epochs, a batch size of 50 and an unlimited number of flips per batch. We also ensured for all of the generated datasets that the DNF concept does not contain more than 20 rules, so that it can theoretically also be learned by the tested shallow network with $s_1 = 20$ (and therefore also for the two deep networks, since their first layer is already bigger).

Figure 6 shows the development of the accuracies on the training set averaged on all 20 datasets over the number of processed mini-batches, whereby after every ten mini-batches a new epoch starts. The base accuracy before processing the first mini-batch and after the full batch optimization are omitted. We can see that the deep networks not only deliver higher accuracies but they also converge slightly faster than the shallow one. The orange curve of DRNC(3) runs a little higher than the blue one of DRNC(5), whereas the green curve of RNC has some distance to them, especially during the first two epochs.

Table 2 shows the accuracies of the three networks. For each dataset, the best accuracy of the three network classifiers is highlighted in bold, the best accuracy including RIPPER and CART in italic. When comparing the rule network approaches, we can see a clear advantage for the two deep networks both when considering the average accuracy and the amount of highest accuracies. The results clearly show that the best performing deep networks outperform the best performing shallow network in all but four of the 20 generated datasets. Both the average rank and the average accuracy of the deep networks is considerably better than the corresponding values for RNC. This also holds for pairwise comparisons of the columns (DRNC(5) vs RNC 15:5, DRNC(3) vs RNC 15:5).

The Friedman test for the ranks yields a significance of more than 95%. A subsequent Nemenyi Test delivers a critical distance of 0.741 (95%) or 0.649 (90%), which shows that DRNC(3) and RNC are significantly different on a level of more than and DRNC(5) and RNC on a level of more than 90%. The corresponding critical distance diagram (CD = 0.741) is shown

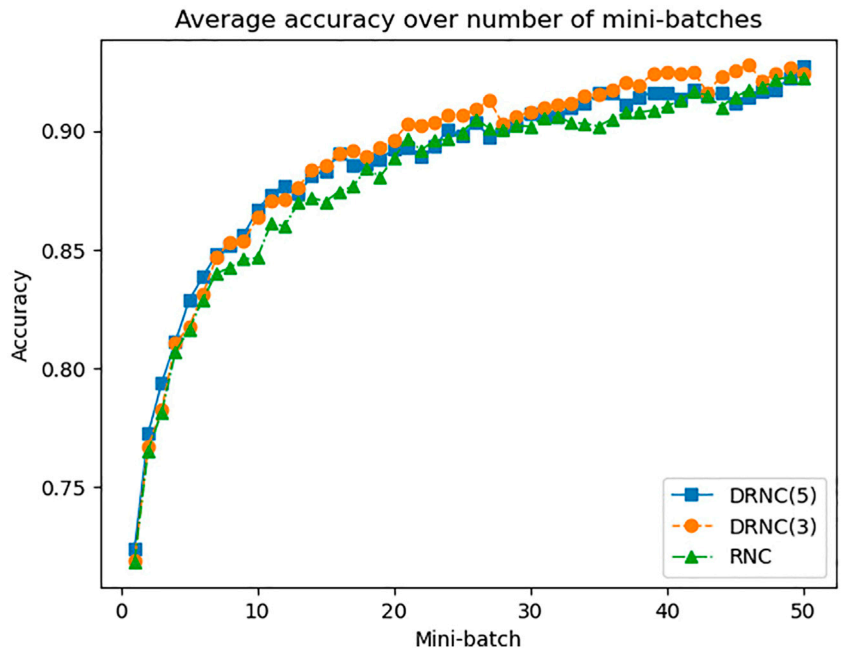

FIGURE 6 | Average accuracy of rule network with 1/3/5 layers on training dataset.

TABLE 2 | Accuracies on artificial datasets. Rule network with 1/3/5 layers vs Ripper vs CART. The best accuracy of the rule networks is marked in bold, the overall best accuracy per dataset is marked in italic.

seed	% (+)	DRNC(5)	DRNC(3)	RNC	Ripper	CART
5	0.4453	0.958	***0.9863***	0.9531	0.9805	0.9844
16	0.7959	0.9639	**0.9707**	0.9629	0.9766	0.9551
19	0.6562	***1***	0.9902	0.9746	1	1
24	0.584	0.9053	0.9043	**0.916**	0.9463	0.9404
36	0.6943	0.8828	***0.9209***	0.9043	0.8867	0.9111
44	0.7939	**0.9629**	0.9551	0.9326	0.9482	0.9697
53	0.6055	**0.9805**	**0.9805**	0.9775	0.9746	0.9824
57	0.7705	**0.9824**	0.9736	0.9639	0.9951	0.9902
60	0.7715	0.9443	**0.9453**	0.9209	0.958	0.9883
65	0.5312	**0.9854**	0.9688	0.9414	0.9961	0.9922
68	0.5654	0.9248	0.9443	**0.9619**	0.9688	0.9355
69	0.6924	0.9551	**0.9658**	0.9199	0.9795	0.9717
70	0.6338	0.9014	0.9062	**0.9229**	0.9111	0.8984
81	0.5684	0.9004	**0.9131**	0.8857	0.9248	0.9756
82	0.7188	0.9941	**0.998**	0.9717	1	1
85	0.5312	***1***	0.998	0.9736	1	1
89	0.6084	0.8926	0.9434	***0.9629***	0.9502	0.9395
107	0.6172	**0.8965**	0.873	0.8643	0.9043	0.9277
112	0.7549	**0.9346**	0.9248	0.9189	0.9082	0.9561
118	0.5957	**0.9688**	0.9414	0.9434	0.9736	0.9688
Ø Accuracy		0.9467	0.9502	0.9386	0.9591	0.9644
Ø Rank		1.775	1.725	2.5	–	–

in **Figure 7**. We thus find it safe to conclude that deep networks outperform shallow networks on these datasets.

In the two right-most columns of **Table 2** we also show a comparison to the state-of-the-art rule learner Ripper (Cohen, 1995) and the decision tree learner CART (Breiman et al., 1984) in Python implementations using default parameters.[4] We see that all network approaches are outperformed by the Ripper and CART classifiers with default setting. The difference between Ripper and DRNC(3) is approximately the same as the difference between DRNC(3) and RNC. However, considering that we only use a naïve greedy algorithm, it could not be expected (and was also not our objective) to be able to beat state-of-the-art rule learner. In particular, the runtime is far from state-of-the-art, since already for the shallow network 30 seconds are needed per dataset and up to 3 minutes for the deep networks (in comparison to less than a second for Ripper and CART). Furthermore, the results also confirm that shallow rule learners (of which both Ripper and CART are representatives) had no disadvantage by the way we generated the datasets.

4.4 Results on UCI Datasets

For an estimation how the rule networks perform on real-world datasets, we select nine classification datasets (*car-evaluation*, *connect-4*, *kr-vs-kp*, *monk* one to three, *mushroom*, *tic-tac-toe* and *vote*) from the UCI Repository (Dua and Graff, 2017). They differ in the number of attributes and instances, but have in common that they consist only of nominal attributes. *Car-evaluation* and *connect-4* are actually multi-class datasets and are therefore converted into the binary classification problem whether a sample belongs to the most frequent class or not. Of all binary classification problems, the networks to be tested treat

[4]We used the implementations available from https://pypi.org/project/wittgenstein/and https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

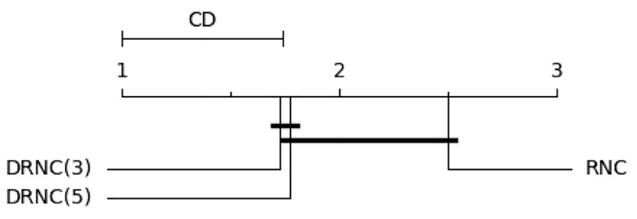

FIGURE 7 | Critical distance diagram for the rule networks on the artificial datasets with a significance level of 95%. Learners that are not statistically different from one another are connected.

TABLE 3 | Accuracies on real-world datasets. Rule network with 1/3/5 layers vs Ripper vs CART. The best accuracy of the rule networks is marked in bold, the overall best accuracy per dataset is marked in italic.

Dataset	% (+)	DRNC(5)	DRNC(3)	RNC	Ripper	CART
car-evaluation	0.7002	0.8999	**0.9022**	0.8565	0.9838	0.9821
connect-4	0.6565	**0.7728**	0.7712	0.7597	0.7475	0.8195
kr-vs-kp	0.5222	0.9671	0.9643	**0.9725**	0.9837	0.989
monk-1	0.5000	***1***	0.9982	0.9910	0.9478	0.8939
monk-2	0.3428	0.7321	**0.7421**	0.7139	0.6872	0.7869
monk-3	0.5199	***0.9693***	0.9603	0.9567	0.9386	0.9729
mushroom	0.784	***1***	0.978	0.993	0.9992	1
tic-tac-toe	0.6534	0.8956	0.9196	**0.9541**	1	0.9217
Vote	0.6138	***0.9655***	0.9288	0.9264	0.9011	0.9287
Ø Rank		1.556	2	2.444	–	–

again the more common class as the positive class and the less common as the negative class, except for the *monk* datasets whereby the positive class is set to 1. As with the artificial datasets, we additionally compare the performance of the networks to Ripper and CART, and again all accuracies are obtained via 2-fold cross validation. In case a random initialization did not yield any result (i.e., the resulting network classified all examples into a single class), we re-initialized with a different seed (this happened once for both deep network versions).

The results are shown in **Table 3**. We can again observe that both deep networks outperform the shallow network RNC. Of all rule networks, DRNC(5) provides the highest accuracy on the *connect-4*, *monk-1*, *monk-3*, *mushroom* and *vote* datasets, whereas DRNC(3) performs best on *car-evaluation* and *monk-2* and RNC on *kr-vs-kp* and *tic-tac-toe*. The comparison to Ripper and CART is again clearly in favor of these state-of-the-art algorithms. We will analyze some of the rule networks in the following subsection.

4.5 Interpretation of Learned Models

While the results in the previous two subsections indicate that deep rule networks perform better than shallow ones on both artificial and UCI datasets, we now try to find the reason for this by analyzing the learned models of all three rule network candidates. We investigate into the models for the artificial dataset generated by seed 44 and the first and third monk

TABLE 4 | DNF models of deep and shallow rule networks for artificial dataset with seed 44. Each row shows a rule (body), which are disjuncted for the final model. Rules that are also contained in the ground truth are marked in bold.

Ground Truth	DRNC(5)	DRNC(3)	RNC
–	–	–	$\neg a \land \neg b \land \neg e \land \neg i \land \neg j$
$\neg b \land c \land \neg i$	–	–	–
$a \land \neg e \land \neg g$	$\boldsymbol{a} \land \neg \boldsymbol{e} \land \neg \boldsymbol{g}$	–	$\neg g$
$\neg b \land \neg g \land i$	$\neg b \land \neg g$	$\neg b \land \neg g$	–
$\neg b \land \neg d \land \neg g$	–	–	–
$f \land \neg g$	$\boldsymbol{f} \land \neg \boldsymbol{g}$	$\boldsymbol{f} \land \neg \boldsymbol{g}$	–
	$\neg g \land \neg h \land \neg j$	–	–
$c \land \neg f \land g$	$\boldsymbol{c} \land \neg \boldsymbol{f} \land \boldsymbol{g}$	$c \land \neg f$	$c \land \neg f$
$c \land \neg f \land \neg j$	–	–	–
	$\neg a \land c \land h$	$c \land h$	$\neg a \land c, c \land h$
$c \land h \land i$	$\boldsymbol{c} \land \boldsymbol{h} \land \boldsymbol{i}$	–	–
$\neg c \land \neg e \land h$	$\neg e \land h$	$\neg e \land h$	$\neg e \land h$
$\neg c \land \neg g \land \neg h$	$\neg \boldsymbol{c} \land \neg \boldsymbol{g} \land \neg \boldsymbol{h}$	$\neg g \land \neg h$	–
$d \land \neg e \land \neg i \land \neg j$	–	$\boldsymbol{d} \land \neg \boldsymbol{e} \land \neg \boldsymbol{i} \land \neg \boldsymbol{j}$	$\boldsymbol{d} \land \neg \boldsymbol{e} \land \neg \boldsymbol{i} \land \neg \boldsymbol{j}$
$\neg a \land f \land \neg i$	$f \land \neg i$	$f \land \neg i$	$f \land \neg i$
$b \land f \land \neg i$	–	–	–
$\neg d \land f \land \neg i$	–	–	–
$f \land \neg h \land \neg i$	–	–	–
$f \land \neg i \land \neg j$	–	–	–

dataset since for those datasets the deep rule networks did not only outperform their shallow counterpart but also Ripper.

For the artificial dataset, we convert the ground truth and the learned networks into equivalent DNF formulas, i.e., flat rules, which allows us to more easily compared the learned concepts with the ground truth. **Table 4** shows the results, where each line corresponds to one rule, with similar rules being grouped together. We notice that all three learned models are smaller than the original one (8–10 rules instead of 16), which is mainly due to the rules in the last five rows, that are combined to a simpler and only marginally worse rule $f \land \neg i \rightarrow \texttt{true}$ by all rule networks. For DRNC(3) and RNC, this is also the case for some other rules, with RNC having the biggest generalization with the simple rule $\neg g \rightarrow$ true. In contrast, DRNC(5) learns more specific rules, and five out of ten are also part of the DNF of the ground truth (highlighted in bold in **Table 4**).

We further investigate into the models learned by the rule networks in terms of interpretability. The shallow network consists of some redundant rules that, however, can easily be converted to the DNF presented in **Table 4**. When looking at the full model of the two deep networks, it is hard to see which concepts of the first layer contribute to the final prediction because of hidden branches and redundancies. After manually simplifying the model of DRNC(3), we notice that the two last layers are not used anymore and we therefore obtain a model similar to the one of the shallow network. While the deep structure seems to promote the finding of a suitable model by offering multiple paths to generate a correct prediction, the resulting model is nevertheless limited to one of these paths and effectively delivers a shallow model that is, however, hard to detect and less interpretable.

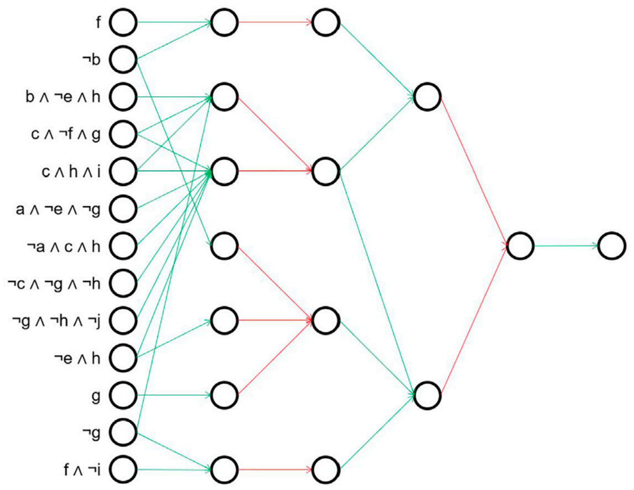

FIGURE 8 | Simplified model of DRNC(5) for artificial dataset with seed 44. For a better overview, the first layer is hidden and instead the resulting rules are listed directly. Red connections are logical ANDs, green edges correspond to logical ORs.

For the model learned by DRNC(5), the simplified structure is still deep. **Figure 8** illustrates this by replacing the first AND-layer with the resulting logical terms, and showing how they are further combined in subsequent layers. In contrast to the model of DRNC(3), even after the removal of redundant concepts and subtrees the structure remains hard to interpret. While some of the underlying concepts in the left-most layer are passed unchanged through the network (e.g. node 4: $c \wedge \neg f \wedge g$), others make actually use of the deep structure and are further combined in subsequent layers (e.g. node 1 and 12 are merged to $f \wedge \neg g$). Some of the aggregations are unnecessarily cumbersome: The concept $\neg e \wedge h$ in node 10 is first conjuncted with nodes 2 and 11, but subsequently disjuncting with nodes 2, 3 and 12 results again in the original concept $\neg e \wedge h$. To conclude, the learned model is reasonable, but not nearly as compact as it could be which requires a deeper analysis to understand the network.

A similar behavior can be recognized for the first monk dataset. Its target concept is $a1 = a2 \vee a5 = 1$, which requires four concepts $a1 = 1 \wedge a2 = 1$, $a1 = 2 \wedge a2 = 2$, $a1 = 3 \wedge a2 = 3$ and $a5 = 1$ to be identified by the tested rule learners. All three rule networks are able to detect these concepts but with minor differences. The shallow rule network RNC learns three of the four concepts directly and uses a combination of two rules for the remaining one. Moreover, the learned model consists of three more rules that are not redundant, but cover additional examples in the test set (but not in the training set). DRNC(3) learns three of the four concepts directly as well. The remaining concept is split into two subconcepts $a1 = 1$ and $a2 = 1$, that are combined in the additional conjunctive layer in the deep structure. As for the RNC model, some additional rules are left that cover a few spurious cases and cause the small error rate during testing. Last, the DRNC(5) model achieves a perfect accuracy by detecting three of the four concepts directly and constructing the remaining one like the DRNC(3) network in the second conjunctive layer.

The results are similar for the third monk dataset with the target concept $a5 = 3 \wedge a4 = 1 \vee a5 \neq 4 \wedge a2 \neq 3$. The inequations have to be learned again by multiple rules, but this time the accuracies are worse for all networks because of the noise that had been added to the data. Both the model of RNC and the simplified model of DRNC(3) are in DNF form, whereas the simplified model of DRNC(5) still makes use of the deep structure. We assume that resulting from this, the DRNC(5) model generalizes better, since in the DRNC(3) and in particular in the RNC model there are additional rules left which overfit the training data. A deep structure seems to provide more and better options to prune the model, since flips in layers close to the output have effects on possibly multiple inputs (this would of course have to be checked in more detail in separate experiments). However, we also note that this accuracy gain comes at the cost of interpretability.

So, in summary, we can observe that useful structures emerge in the deep networks, which are, however, not easily interpretable. We are currently working on employing methods for logic minimization to reduce the size of a learned deep network in order to increase their interpretability.

5 CONCLUSION

The main objective of this work was to study the question whether deep rule networks have the potential of outperforming shallow DNF rule sets, even though, in principle, every concept can be represented as DNF formula. As there is no sufficiently competitive deep rule learning algorithm, we proposed a technique how deep and shallow rule networks can be learned and thus effectively compared in a uniform framework, using a network approach with a greedy, mini-batch based optimization algorithm. For both types of networks, we find good hyperparameter settings that allow the networks to reach reasonable accuracies on both artificial and real-world datasets, even though the approach is still outperformed by state-of-the-art learning algorithms such as Ripper and CART.

Our experiments on both artificial and real-world benchmark data indicate that deep rule networks outperform shallow networks. The deep networks obtain not only a higher accuracy, but also need less mini-batch iterations to achieve it. Moreover, in preliminary experiments in the hyperparameter grid search, we have seen indications that the deep networks are generally more robust to the choice of the hyperparameters than shallow networks. On the other hand, we also had some cases on real-world data sets where deep networks failed because a poor initialization resulted in indiscriminate predictions. The current approach is also limited to binary classification problems with nominal attributes.

Overall, we interpret these results as evidence that an investigation of deep rule structures is a promising research goal, which we hope could yield a similar boost in performance in inductive rule learning as could be observed by moving from shallow to deep neural networks. However, this goal is still far ahead.

6 FUTURE WORK

In this work, it was not our goal to reach a state-of-the-art predictive performance, but instead we wanted to evaluate a very simple greedy optimization algorithm on both shallow and deep networks, in order to get an indication on the potential of deep rule networks. Nevertheless, several avenues for improving our networks have surfaced, which we intend to explore in the near future.

One of the main drawbacks of the presented deep rule networks is the extremely high runtime due to the primitive flipping algorithm. A single flip needs a recalculation of all activations in the network, even if only a few them will be affected by this flip whereby the matrix multiplication could be minimized considerably. Conversely, this knowledge can be used to find a small subset of flips that affects a certain activation. On the other hand, the majority of possible flips does not have any effect on this activation or the accuracy at all. This effect will typically remain unchanged after a few more flips are done. Therefore, an exhaustive search of all flips is only needed in the first iteration, while afterwards just a subset of possible flips should be considered which can be built either in a deterministic or probabilistic way.

Due to this lack of backpropagation, the flips are evaluated by their influence on the prediction when executed. However, when looking at a false positive, we can only correct this error by making the overall hypothesis of the network more specific. In order to achieve a generalization of the hypothesis, only flips from *false* to `true` in conjunctive layers or flips from `true` to false in disjunctive layers have to be taken into account. In this way, all flips are split into "generalization-flips" and "specialization-flips" of which only one group has to be considered at the same time. This improvement as well as the above-mentioned selection of a subset of flips might also allow us to perform two or more flips at the same time so that a better result than with the greedy approach can be achieved.

An even more promising approach starts one step earlier in the initialization phase of the network. Instead of specifying the structure of the network and finding optimal initialization parameters $\bar{l}$ and p for it, a small part of the data could be used to create a rough draft version of the network. The Quine-McCluskey algorithm (McCluskey, 1956), Ripper and the espresso-algorithm (Brayton et al., 1984) are suitable methods to generate shallow networks, whereas decision trees and decision graphs can be used to generate deep networks since the contained rules already share some conditions and, moreover, similar subtrees can be merged.

All these approaches share some significant advantages over the network approach we developed so far. First of all, the decision which class value will be treated as positive or negative does not have to be made manually any longer. Second, they automatically deliver a suitable initialization of the network, which otherwise would have to be improved by similar approaches like used in neural networks (e.g., Ramos et al., 2017) to achieve a robust performance. Third, the general structure of the network is not limited to a fixed size and depth where each node is strictly assigned to a specific layer. Instead of generating nodes that become useless after a few flips have been processed and that should be removed, we can thereby start with a small structure which can be adapted purposefully by copying and mutating good nodes and pruning bad ones. However, it remains unclear whether these changes still lead to improvements in performance or if the network in the given structure is already optimal.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in this online repository: https://github.com/f-beck/scikit-learn-rule-network.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

ACKNOWLEDGMENTS

Supported by Johannes Kepler Open Access Publishing Fund. We are grateful to Eneldo Loza Mencía, Michael Rapp, Eyke Hüllermeier, and Van Quoc Phuong Huynh for inspiring discussions, fruitful pointers to related work, and for suggesting the evaluation with artificial datasets. Thanks are also due to the reviewers for their encouraging comments and concrete suggestions for improving this paper.

REFERENCES

Andrews, R., Diederich, J., and Tickle, A. B. (1995). Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. *Knowledge-Based Syst.* 8, 373–389. doi:10.1016/0950-7051(96)81920-4

Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M. I., and Rudin, C. (2017). Learning Certifiably Optimal Rule Lists for Categorical Data. *J. Machine Learn. Res.* 18, 234:1–234:78. https://jmlr.org/papers/v18/17-716.html.

Barakat, N., and Bradley, A. P. (2010). Rule Extraction from Support Vector Machines: A Review. *Neurocomputing* 74, 178–190. doi:10.1016/j.neucom.2010.02.016

Beck, F., and Fürnkranz, J. (2020). An Investigation into Mini-Batch Rule Learning in Proceedings of the 2nd Workshop on Deep Continuous-Discrete Machine Learning (DeCoDeML), Ghent, Belgium Editors K. Kersting, S. Kramer, and Z. Ahmadi

Brayton, R. K., Hachtel, G. D., McMullen, C. T., and Sangiovanni-Vincentelli, A. L. (1984). "Logic Minimization Algorithms for VLSI Synthesis," in Logic Minimization Algorithms for VLSI Synthesis, vol. 2 of The Kluwer International Series in Engineering and Computer Science (Springer). doi:10.1007/978-1-4613-2821-6

Breiman, L., Friedman, J. H., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Brooks).

Breiman, L. (2001). Random Forests. *Machine Learn.* 45, 5–32. doi:10.1023/a:1010933404324

Burkhardt, S., and Kramer, S. (2015). "On the Spectrum between Binary Relevance and Classifier Chains in M Ulti-Label Classification," in Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC), Editors R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong (Salamanca, Spain: ACM), 885–892. doi:10.1145/2695664.2695854

Cohen, W. W. (1995). "Fast Effective Rule Induction," in Proceedings of the 12th International Conference on Machine Learning (ML-95). Editors A. Prieditis and S. Russell (Lake Tahoe, CA: Morgan Kaufmann), 115–123. doi:10.1016/b978-1-55860-377-6.50023-2

Cohen, W., Yang, F., and Rivard Mazaitis, K. (2020). TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *jair* 67, 285–325. doi:10.1613/jair.1.11944

Courbariaux, M., Bengio, Y., and David, J. (2015). "Binaryconnect: Training Deep Neural Networks with Binary Weights during Propagations," in *Advances in Neural Information Processing Systems 28 (NIPS)*. Editors C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Montreal, Quebec: Canada), 3123–3131.

Craven, M. W., and Shavlik, J. W. (1997). Using Neural Networks for Data Mining. *Future Generation Comput. Syst.* 13, 211–229. doi:10.1016/S0167-739X(97)00022-8

Cristianini, N., and Shawe-Taylor, J (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press. 1st Edition, 211

De Raedt, L., Lavrač, N., and Džeroski, S. (1993). "Multiple Predicate Learning," in Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambéry, France. Editor R. Bajcsy (Chambéry, France: Morgan Kaufmann), 1037–1043.

Delalleau, O., and Bengio, Y. (2011). "Shallow vs. Deep Sum-Product Networks," in *Advances in Neural Information Processing Systems 24 (NIPS)*. Editors J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger (Granada, Spain), 666–674.https://proceedings.neurips.cc/paper/2011.

Dembczyński, K., Kotłowski, W., and Słowiński, R. (2010). ENDER: a Statistical Framework for Boosting Decision Rules. *Data Min Knowl Disc* 21, 52–90. doi:10.1007/s10618-010-0177-7

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On Label Dependence and Loss Minimization in Multi-Label Classification. *Machine Learn.* 88, 5–45.

[Dataset] Dua, D., and Graff, C. (2017). *UCI Machine Learning Repository*. https://archive.ics.uci.edu.

Evans, R., and Grefenstette, E. (2018). Learning Explanatory Rules from Noisy Data. *jair* 61, 1–64. doi:10.1613/jair.5714

Friedman, J. H., and Popescu, B. E. (2008). Predictive Learning via Rule Ensembles. *Ann. Appl. Stat.* 2, 916—-954. doi:10.1214/07-AOAS148

Fürnkranz, J. (2005). "From Local to Global Patterns: Evaluation Issues in Rule Learning Algorithms," in *Local Pattern Detection*. Editors K. Morik, J.-F. Boulicaut, and A. Siebes (Springer-Verlag), 20–38. doi:10.1007/11504245_2

Fürnkranz, J., Gamberger, D., and Lavrač, N. (2012). *Foundations of Rule Learning*. Springer-Verlag.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., and Rapp, M. (2020). "Learning Structured Declarative Rule Sets – a challenge for Deep Discrete Learning," in Proceedings of the 2nd Workshop on Deep Continuous-Discrete Machine Learning (DeCoDeML), Ghent, Belgium. Editors K. Kersting, S. Kramer, and Z. Ahmadi.

González, C., Loza Mencía, E., and Fürnkranz, J. (2017). "Re-training Deep Neural Networks to Facilitate Boolean Concept Extraction," in Proceedings of the 20th International Conference on Discovery Science (DS-17), Kyoto, Japan (Springer-Verlag), 127–143. vol. 10558 of Lecture Notes in Computer Science. doi:10.1007/978-3-319-67786-6/TNQDotTNQ/1010.1007/978-3-319-67786-6_10

Guerreiro, J., and Trigueiros, D. (2010). "A Unified Approach to the Extraction of Rules from Artificial Neural Networks and Support Vector Machines," in Proceedings of the 6th International Conference on Advanced Data Mining and Applications (ADMA), Part II, Chongqing, China. Editors L. Cao, J. Zhong, and Y. Feng (Chongqing, China: Springer), 34–42. doi:10.1007/978-3-642-17313-4_4

Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., and Giannotti, F. (2018). *Local Rule-Based Explanations of Black Box Decision Systems*. arXiv Preprint 1805.10820. https://arxiv.org/abs/1805.10820.

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learn.* 11, 63–9010. doi:10.1023/a:1022631118932

Hornik, K. (1991). Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks* 4, 251–257. doi:10.1016/0893-6080(91)90009-T

Hüllermeier, E., Fürnkranz, J., Loza Mencia, E., Nguyen, V.-L., and Rapp, M. (2020). "Rule-based Multi-Label Classification: Challenges and Opportunities," in Proceedings of the 4th International Joint Conference on Rules and Reasoning (RuleML+RR). Editors V. Gutiérrez-Basulto, T. Kliegr, A. Soylu, M. Giese, and D. Roman (Oslo, Norway: Springer), 3–19. vol. 12173 of Lecture Notes in Computer Science. doi:10.1007/978-3-030-57977-7_1

Kijsirikul, B., Numao, M., and Shimura, M. (1992). "Discrimination-based Constructive Induction of Logic Programs," in Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, USA, 44–49.

Kok, S., and Domingos, P. M. (2007). "Statistical Predicate Invention," in Proceedings of the 24th International Conference on Machine Learning (ICML-07), Corvallis, Oregon, USA. Editor Z. Ghahramani (Corvallis, Oregon, USA: ACM), 433–440. vol. 227 of ACM International Conference Proceeding Series. doi:10.1145/1273496.1273551

Kramer, S. (2020). "A Brief History of Learning Symbolic Higher-Level Representations from Data (And a Curious Look Forward)," in Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI), Survey Track, Yokohama, Japan, 4868–4876. doi:10.24963/ijcai.2020/678

Lakkaraju, H., Bach, S. H., and Leskovec, J. (2016). "Interpretable Decision Sets," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-16). Editors B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi (San Francisco, CA: ACM)), 1675–1684. doi:10.1145/2939672.2939874

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep Learning. *Nature* 521, 436–444. doi:10.1038/nature14539

Li, F., Zhang, B., and Liu, B. (2016). *Ternary Weight Networks*. arxiv abs/1605.04711. https://arxiv.org/abs/1605.04711.

Malerba, D., Semeraro, G., and Esposito, F. (1997). "A Multistrategy Approach to Learning Multiple Dependent Concepts," in *Machine Learning and Statistics: The Interface*. Editors G. Nakhaeizadeh and C. C. Taylor (London, England: Wiley), 4, 87–106.

Matheus, C. J. (1989). "A Constructive Induction Framework," in Proceedings of the 6th International Workshop on Machine Learning, Ithaca, NY, USA, 474–475. doi:10.1016/b978-1-55860-036-2.50121-1

McCluskey, E. J. (1956). Minimization of Boolean Functions*. *Bell Syst. Tech. J.* 35, 1417–1444. doi:10.1002/j.1538-7305.1956.tb03835.x

Mhaskar, H., Liao, Q., and Poggio, T. A. (2017). "When and Why Are Deep Networks Better Than Shallow Ones?," in Proceedings of the 31st AAAI Conference on Artificial Intelligence, San Francisco, California, USA. Editors S. P. Singh and S. Markovitch (San Francisco, California, USA: AAAI Press), 2343–2349.

Michalski, R. S. (1969). "On the Quasi-Minimal Solution of the Covering Problem," in Proceedings of the 5th International Symposium on Information Processing (FCIP-69), Yugoslavia (Yugoslavia: Bled: Switching Circuits), A3, 125–128.

Morik, K., Wrobel, S., Kietz, J.-U., and Emde, W. (1993). *Knowledge Acquisition and Machine Learning – Theory, Methods, and Applications*. London: Academic Press.

Muggleton, S., and Buntine, W. (1988). "Machine Invention of First-Order Predicates by Inverting Resolution," in Proceedings of the 5th International Conference on Machine Learning (ML-88), Ann Arbor, MI, USA, 339–352. doi:10.1016/b978-0-934613-64-4.50040-2

Muggleton, S. H., Lin, D., and Tamaddoni-Nezhad, A. (2015). Meta-interpretive Learning of Higher-Order Dyadic Datalog: Predicate Invention Revisited. *Mach Learn.* 100, 49–73. doi:10.1007/s10994-014-5471-y

Muggleton, S. H. (1987). "Structuring Knowledge by Asking Questions," in *Progress in Machine Learning*. Editors I. Bratko and N. Lavrač (Wilmslow, England: Sigma Press), 218–229.

Nam, J., Loza Mencía, E., and Fürnkranz, J. (2016). "All-in Text: Learning Document, Label, and Word Representations Jointly," in Proceedings of the 30th AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA. Editors D. Schuurmans and M. P. Wellman (AAAI Press), 1948–1954. https://aaai.org/Library/AAAI/aaai16contents.php.

Pfahringer, B. (1994). "Controlling Constructive Induction in CiPF: an MDL Approach," in Proceedings of the 7th European Conference on Machine Learning (ECML-94), Catania, Sicily. Editor P. B. Brazdil (Catania, Sicily:

Springer-Verlag), 242–256. Lecture Notes in Artificial Intelligence. doi:10.1007/3-540-57868-4_62

Polato, M., and Aiolli, F. (2019). Boolean Kernels for Rule Based Interpretation of Support Vector Machines. *Neurocomputing* 342, 113–124. doi:10.1016/j.neucom.2018.11.094

Poon, H., and Domingos, P. M. (2011). "Sum-product Networks: A New Deep Architecture," in Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI), Barcelona, Spain. Editors F. G. Cozman and A. Pfeffer (Barcelona, Spain: AUAI Press), 337–346. doi:10.1109/iccvw.2011.6130310

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. (2020). "Binary Neural Networks: A Survey," in *Pattern Recognitition*. Elsevier, 105, 107281. doi:10.1016/j.patcog.2020.107281

Ramos, E. Z., Nakakuni, M., and Yfantis, E. (2017). "Quantitative Measures to Evaluate Neural Network Weight Initialization Strategies," in IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017, Las Vegas, NV, USA, January 9-11, 2017 (IEEE), 1–7. doi:10.1109/CCWC.2017.7868389

Rapp, M., Loza Mencía, E., Fürnkranz, J., Nguyen, V.-L., and Hüllermeier, E. (2020). "Learning Gradient Boosted Multi-Label Classification Rules," in Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), Part III. Editors F. Hutter, K. Kersting, J. Lijffijt, and I. Valera (Springer-Verlag), 124–140. vol. 12459 of Lecture Notes in Computer Science.

Read, J., and Hollmén, J. (2014). "A Deep Interpretation of Classifier Chains," in *Advances in Intelligent Data Analysis 13 (IDA)*. Editors H. Blockeel, M. van Leeuwen, and V. Vinciotti (Leuven, Belgium: Springer), 251–262. vol. 8819 of Lecture Notes in Computer Science. doi:10.1007/978-3-319-12571-8_22

Read, J., and Hollmén, J. (2015). *Multi-label Classification Using Labels as Hidden Nodes*. Available at: https://arxiv.org/abs/1503.09022. CoRR abs/1503.09022.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier Chains for Multi-Label Classification. *Mach Learn.* 85, 333–359. doi:10.1007/s10994-011-5256-5

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2021). Classifier Chains: A Review and Perspectives. *jair* 70, 683–718. doi:10.1613/jair.1.12376

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). ""Why Should I Trust You?"," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Francisco, CA, USA. Editors B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi (San Francisco, CA, USA: ACM), 1135–1144. doi:10.1145/2939672.2939778

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks* 61, 85–117. doi:10.1016/j.neunet.2014.09.003

Schmitz, G. P. J., Aldrich, C., and Gouws, F. S. (1999). ANN-DT: an Algorithm for Extraction of Decision Trees from Artificial Neural Networks. *IEEE Trans. Neural Netw.* 10, 1392–1401. doi:10.1109/72.809084

Senge, R., and Hüllermeier, E. (2011). Top-down Induction of Fuzzy Pattern Trees. *IEEE Trans. Fuzzy Syst.* 19, 241–252. doi:10.1109/TFUZZ.2010.2093532

Setzu, M., Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. (2021). Glocalx-From Local to Global Explanations of Black Box AI Models. *Artif. Intell.* 294, 103457. doi:10.1016/j.artint.2021.103457

Sommer, E. (1996). *Theory Restructuring – A Perspective on Design and Maintenance of Knowlege Based Systems, Vol. 171 of DISKI (Infix)*, St. Augustin, Germany: Infix-Verlag.

Stahl, I. (1996). "Predicate Invention in Inductive Logic Programming," in *Advances in Inductive Logic Programming*. Editor L. De Raedt (IOS Press), 34–47. vol. 32 of Frontiers in Artificial Intelligence and Applications. https://www.iospress.com/catalog/books/advances-in-inductive-logic-programming.

Tsoumakas, G., and Katakis, I. (2007). Multi-Label Classification. *Int. J. Data Warehousing Mining* 3, 1–1310. doi:10.4018/jdwm.2007070101

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). "Mining Multi-Label Data," in *Data Mining and Knowledge Discovery Handbook*. Editors O. Maimon and L. Rokach. 2nd edn. (Springer), 667–685. doi:10.1007/978-0-387-09823-4_34

Waegeman, W., Dembczyński, K., and Hüllermeier, E. (2019). Multi-target Prediction: a Unifying View on Problems and Methods. *Data Min Knowl Disc* 33, 293–324. doi:10.1007/s10618-018-0595-5

Wang, T., Rudin, C., Doshi-Velez, F., Liu, Y., Klampfl, E., and MacNeille, P. (2017). A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *J. Machine Learn. Res.* 18, 70:1–70:37.

Wnek, J., and Michalski, R. S. (1994). Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learn.* 14, 139–168. Special Issue on Evaluating and Changing Representation. doi:10.1023/a:1022622132310

Zhang, M.-L., and Zhou, Z.-H. (2014). A Review on Multi-Label Learning Algorithms. *IEEE Trans. Knowl. Data Eng.* 26, 1819–1837. doi:10.1109/tkde.2013.39

Zhu, C., Han, S., Mao, H., and Dally, W. J. (2017). "Trained Ternary Quantization," in Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France (Toulon, France: OpenReview.net).

Zilke, J. R., Loza Mencia, E., and Janssen, F. (2016). "DEEPRED-Rule Extraction From Deep Neural Networks," in Proceedings of the 19th International Conference on Discovery Science (DS-16). Editors T. Calders, M. Ceci, and D. Malerba (Bari, Italy: Springer International Publishing), 457–473. doi:10.1007/978-3-319-46307-029

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Beck and Fürnkranz. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.