Beyond DNF: First Steps towards Deep Rule Learning

Florian Beck[1], Johannes Fürnkranz[1]

Institute for Application-oriented Knowledge Processing (FAW)
Department of Computer Science
Johannes Kepler University Linz, Austria
{fbeck,juffi}@faw.jku.at

Abstract: 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. 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. In this paper, we investigate into networks with weights and activations limited to the values 0 and 1. For the lack of a powerful algorithm that optimizes deep rule sets, we empirically compare deep and shallow rule networks with a uniform general algorithm, which relies on greedy mini-batch based optimization. Our experiments on both artificial and real-world benchmark data indicate that deep rule networks may outperform shallow networks.

1 Introduction

Dating back to the AQ algorithm [13], inductive rule learning is one of the most traditional fields in machine learning. However, when reflecting upon its long history [7], it can be argued that while modern methods are somewhat more scalable than traditional rule learning algorithms [see, e.g., 16, 10], no major break-through has been made. In fact, the RIPPER rule learning algorithm [5] 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 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 [12]. 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 [9]. 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. We also test on both, real-world UCI benchmark datasets, as well as artificial datasets for which we know the underlying target concept representations.

The remainder of the paper is organized as follows: Sect. 2 elaborates why deep rule learning is of particular interest and refers to related work. We propose a new network approach in Sect. 3 and test it in Sect. 4. The results are concluded in Sect. 5, followed by possible future extensions and improvements in Sect. 6. An extended version of this paper containing an elaborate discussion of related work, more details on the methods, and additional experiments is available as [2].

2 Deep Rule Learning

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 $2 \cdot (R-1) = 8$

[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.

```
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.
```

(a) A flat unstructured rule set for the parity concept

```
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.
```

(b) A deep structured rule base for parity using three auxiliary predicates

Figure 1: Unstructured and structured rule sets for the parity concept.

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 5 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 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.

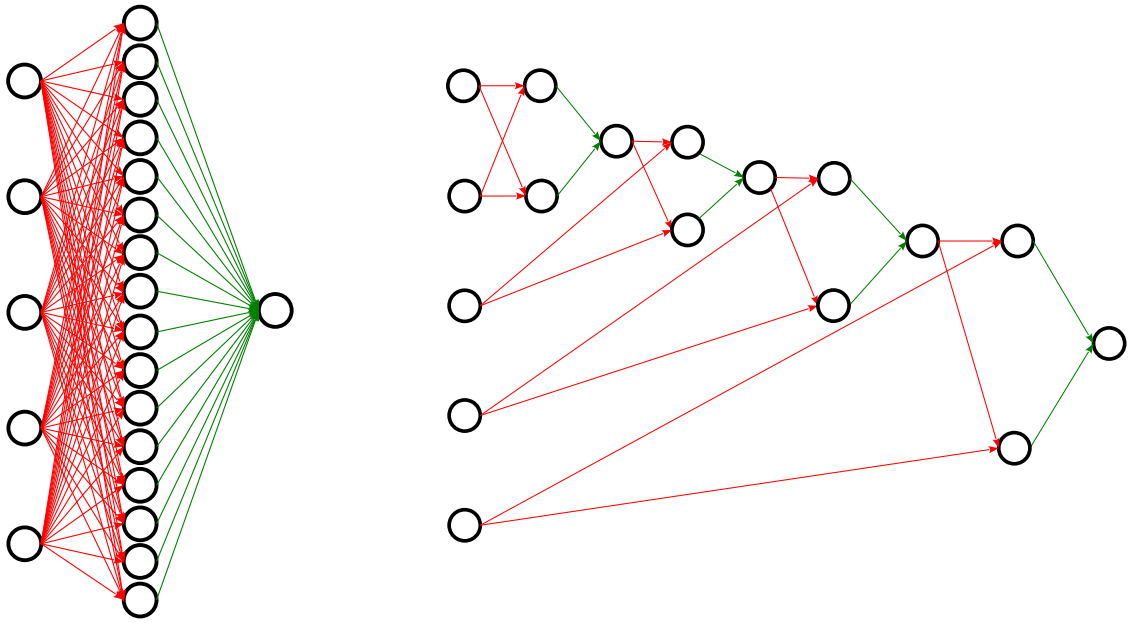
It is important to note that deep structures do not increase the expressiveness of the learned concepts. Any formula in propositional logic (and we limit ourselves to propositional logic in this project) can be converted to a DNF formula. In the worst case (a so-called *full DNF*), each of the input 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 [9], 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 [12]. 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.

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 we have previously used for experimenting with mini-batch rule learning [1], 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



(a) shallow representation (b) deep representation

Figure 2: Network representations of the parity rule sets of Figure 1. Red connections are logical ANDs, green edges correspond to logical ORs.

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. Note that we can still emulate a shallow DNF-structure by choosing a single hidden layer. In the more general case, the hidden layers are treated alternately as conjunctive and disjunctive layers. We focus on layer structures starting with a conjunctive hidden layer and ending with a disjunctive output layer, i.e. networks with an odd number of hidden layers. 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 $s_0 = |\mathbf{x}|$ 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 $i \bmod 2 = 0$) or disjunction (if $i \bmod 2 = 1$) 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 1 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 $|\mathscr{A}|$ be the number of attributes, then each attribute is selected with the probability $\bar{l}/|\mathscr{A}|$ 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 ($\odot$). In each layer i, the input features $A^{(i)}$ are multiplied with the corresponding weights $W^{(i)}$ and aggregated at the receiving node in layer $i+1$. If the aggregation is disjunctive, this directly corresponds to a binary matrix multiplication. 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

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

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 ($A^{(0)} = \mathbf{x}$) 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.

3.4 Training

Following [1], 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. 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.

4.1 Artificial Datasets

As many standard UCI databases can be solved with very simple rules [8], 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 ensured in preliminary experiments).

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 $\mathbf{s} = [32, 16, 8, 4, 2]$. 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.

4.2 Results on Artificial Datasets

We first conducted a few preliminary experiments on three of the artificial datasets to set suitable default values for the hyperparameters of the deep and shallow networks. The detailed grid search including figures comparing different hyperparameter settings are presented in the extended version of this paper [2]. Based on these results, we selected three network versions for the main experiments. 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 $\mathbf{s} = [32, 16, 8, 4, 2]$ 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: $\mathbf{s} = [32, 8, 2]$ 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 $\mathbf{s} = [32, 16, 8, 4, 2]$, DRNC(3) for $\mathbf{s} = [32, 8, 2]$ 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.

In the main experiments, we use a combination of 15 artificial datasets with seeds we already used in the prior hyperparameter grid search and 5 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 be 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).

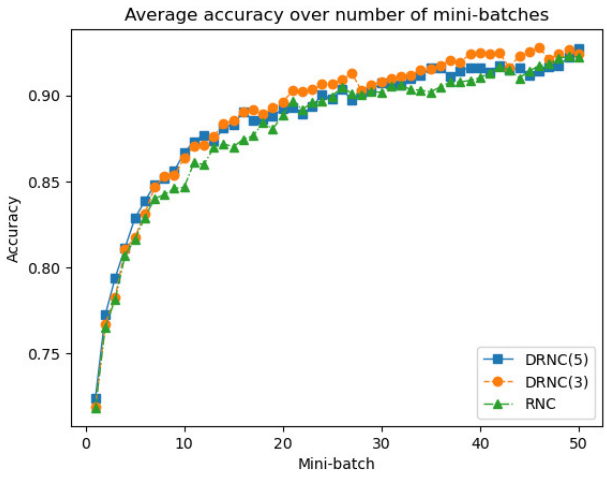


Figure 3: Average accuracy of rule network with 1/3/5 layers on training dataset.

Figure 3 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 1 shows the accuracies of the three networks. For each dataset, the best accuracy of the three network classifiers is highlighted in bold. 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 4 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).

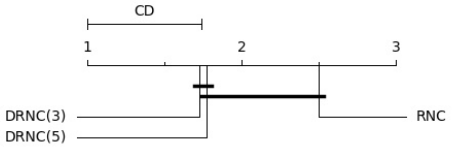


Figure 4: Critical distance diagram for the rule networks on the artificial datasets with a significance level of 95%. Learners that are not statistically different from one another are connected.

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 95% and DRNC(5) and RNC on a level of more than 90%. The corresponding critical distance diagram (CD=0.741) is shown in Figure 4. We thus find it safe to conclude that deep networks outperform shallow networks on these datasets.

In the two right-most columns of Table 1 we also show a comparison to the state-of-the-art rule learner RIPPER [5] and the decision tree learner CART [4] in Python implementations using default parameters.[2] 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

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

Table 1: Accuracies on artificial datasets. Rule network with 1/3/5 layers vs RIPPERvs 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		

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 three 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.

4.3 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* 1-3, *mushroom*, *tic-tac-toe* and *vote*) from the UCI Repository [6]. 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 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 2. We can again observe that the deep 5-layer network DRNC(5) outperforms 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 latter two datasets are also interesting: *tic-tac-toe* clearly does not require a deep structure, because for solving it, the learner essentially needs to enumerate all three-in-a-row positions on a 3×3 board. This is similar to *connect-4*, where four-in-a-row positions have to be recognized. However, in the former case, there is only one matching tile for an intermediate concept consisting of two tiles, while in *connect-4* there are several, which can potentially be exploited by a deeper network. In the *kr-vs-kp* dataset, deep structures are also not helpful because it consists of carefully engineered features for the KRKP chess endgame, which were designed in an iterative process so that the game can be learned with a decision tree learner [15]. It would be an ambitious goal of deep rule learning methods to be able to learn such a dataset from, e.g., only the positions of the chess pieces. This is clearly beyond the state-of-the-art of current rule learning algorithms. The comparison to RIPPER and CART is again clearly in favor of these state-of-the-art algorithms.

Table 2: 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		

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 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.

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 [11] or RIPPER are suitable methods to generate shallow networks, whereas the `ESPRESSO`-algorithm [3] would generate deep networks.

Decision trees can also 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., 14] 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.

Acknowledgments. 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.

References

[1] F. Beck and J. Fürnkranz. An investigation into mini-batch rule learning. In K. Kersting, S. Kramer, and Z. Ahmadi, editors, *Proceedings of the 2nd Workshop on Deep Continuous-Discrete Machine Learning (DeCoDeML)*, 2020. Available as arXiv Preprint abs/2106.10202.

[2] F. Beck and J. Fürnkranz. An empirical investigation into deep and shallow rule learning. *arxiv Preprint*, abs/2106.10254, 2021. Submitted for journal publication.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science*. Springer, 1984.

[4] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA, 1984.

[5] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning (ML-95)*, pages 115–123, Lake Tahoe, CA, 1995. Morgan Kaufmann.

[6] D. Dua and C. Graff. UCI machine learning repository, 2017.

[7] J. Fürnkranz, D. Gamberger, and N. Lavrač. *Foundations of Rule Learning*. Springer-Verlag, 2012.

[8] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

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

[10] H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-16)*, pages 1675–1684, San Francisco, CA, 2016. ACM.

[11] E. J. McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[12] H. Mhaskar, Q. Liao, and T. A. Poggio. When and why are deep networks better than shallow ones? In S. P. Singh and S. Markovitch, editors, *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2343–2349, San Francisco, California, USA, 2017. AAAI Press.

[13] R. S. Michalski. On the quasi-minimal solution of the covering problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, volume A3 (Switching Circuits), pages 125–128, Bled, Yugoslavia, 1969.

[14] E. Z. Ramos, M. Nakakuni, and E. Yfantis. Quantitative measures to evaluate neural network weight initialization strategies. In *Proceedings of the IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–7, Las Vegas, NV, USA, 2017. IEEE.

[15] A. D. Shapiro. *Structured Induction in Expert Systems*. Turing Institute Press. Addison-Wesley, 1987.

[16] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille. A Bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 18:70:1–70:37, 2017.