Towards SAT-Based Learning of NNF Networks

Paul Seip, Florian Beck, Johannes Fürnkranz
Institute for Application-Oriented Knowledge Processing
Johannes Kepler University Linz, Austria
{first.last}@jku.at

Clemens Hofstadler, Peter Pfeiffer, Martina Seidl
Institute for Symbolic Artificial Intelligence
Johannes Kepler University Linz, Austria
{first.last}@jku.at

Robert Peharz
Institute of Machine Learning and Neural Computation
TU Graz, Austria
Robert.peharz@tugraz.at

Stefan Szeider
Algorithms and Complexity Group
TU Wien, Austria
sz@ac.tuwien.ac.at

***Abstract*—To solve classification problems, we present a novel SAT-based framework for learning NNF networks directly from binary input-output data. In analogy to deep neural networks, negation normal form (NNF) networks exhibit a deep structure with learnable Boolean weights. By encoding this learning problem as a propositional satisfiability instance, our method leverages modern SAT solvers to construct logically correct, compact, and interpretable Boolean models of given datasets. Unlike statistical learners or heuristic rule induction algorithms, our approach guarantees logical fidelity on the training data. Evaluations on synthetic benchmarks and real-world datasets demonstrate that it outperforms state-of-the-art rule-based algorithms as well as a greedy NNF learning algorithm in terms of accuracy.**

***Index Terms*—Classification, Rule learning, SAT solving**

I. Introduction

We introduce an exact SAT-based approach for learning *negation normal form (NNF) networks* based on training data given as truth tables. The motivation for training NNF networks to solve classification problems lies in the realization that the alternating layers of conjunctive and disjunctive nodes may be interpreted as deeply structured propositional rule sets. The ability to learn such deep structures distinguishes our work from common rule learning algorithms such as the traditional RIPPER and more modern algorithms such as CORELS [1] or LORD [2], which are only able to learn flat expressions in disjunctive normal form (DNF).

Our method formulates the learning problem as propositional formulas, leveraging the remarkable advancements in modern SAT solving [3]. The central idea is to define a network template that consists of alternating conjunctive and disjunctive layers with *Boolean weights* on the edges to determine whether a connection is active or inactive, analogous to neural networks. We then express its behavior symbolically and use a SAT solver to learn the corresponding weights of the network. This allows us to find deep rule sets that exactly encode a dataset within the bounds of a given structure (if such a representation exists). Our results show that SAT solvers can be effectively harnessed for constructing logical structures from data. This opens new possibilities for combining symbolic reasoning with data-driven learning, and contributes to the broader goal of interpretable and verifiable AI systems [4].

II. Related Work

The idea of encoding rule learning problems as satisfiability problems has been considered, e.g., for rule sets and decision lists [5], as well as for decision trees [6]. Similarly, other discrete optimization algorithms, such as (mixed) integer linear programming, have been employed. For example, [7] and [8] optimized an integer programming approach for rule learning using column generation, which addresses the problem that the number of possible candidate rules that need to be evaluated is exponential in the number of input features. In [1], a custom discrete optimization algorithm is proposed that can find exact loss minimizing rule lists. However, all these works are limited to learning flat rule sets or decision lists.

To our knowledge, this is the first method that explicitly targets exact NNF synthesis using satisfiability encodings, offering a principled approach to logic learning that aligns with the goals of knowledge compilation and symbolic AI. It is most similar to [9], which also aimed at learning a layered representation of Boolean networks with alternating conjunctive and disjunctive layers, drawing inspiration from analogy to deep neural networks: flat NNFs (in particular DNFs) can, in principle, represent every Boolean function, in the same way as shallow neural networks with a single hidden layer can represent an arbitrary continuous function by the universal approximation theorem [10]. However, deep neural networks are usually easier to train and often yield better performance, presumably because they require exponentially fewer parameters than shallow networks [11]. The authors investigated a stochastic hill-climbing algorithm for optimizing the weights in an NNF network (which they call a deep rule network), and demonstrated its potential on both artificial and real-world datasets. Consequently, we use this greedy algorithm as a baseline for this work.

III. Negation Normal Form Networks

Propositional formulas and their semantics are defined as usual. Formulas in *negation normal form* (NNF) [12] only use conjunctions and disjunctions as binary connectives. Negation is only allowed in front of atoms. A NNF is called positive if it does not contain any negation symbols.

Example 1. *The following formula is a positive NNF:*

$$(a \vee b) \wedge (c \vee d) \wedge \big(e \vee (f \wedge g)\big) \tag{1}$$

Special forms of NNFs are fomulas in conjunctive normal form (CNF), i.e., conjunctions of disjunctions of literals (a variable or a negated variable) and formulas in disjunctive normal form, i.e., disjunctions of conjunctions of literals.

Given a training dataset of examples, we will use propositional logic to automatically find classification rules that classify unseen instances. We assume a tabular training dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where each example $\mathbf{x}_i$ consists of d Boolean values $\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,d}$. If $\mathbf{x}_{i,j} = 1$, then feature $f_j = 1$ and if $\mathbf{x}_{i,j} = 0$, then $f_j = 0$ for $\mathbf{x}_i$. We may view $\mathcal{D}$ as the binary matrix $\mathcal{D} = (\mathbf{x}_{i,j}) \in \{0,1\}^{n\times d}$. Each row of $\mathcal{D}$ corresponds to one example and each column corresponds to the values of one feature. The training data is labeled, that is, there is a n-dimensional vector $\mathbf{y} \in \{0,1\}^n$ that assigns a Boolean value y_i to each example $\mathbf{x}_i$. Example $\mathbf{x}_i$ is called *positive* if $y_i = 1$, and *negative* if $y_i = 0$.

Example 2. *Consider a collaborative task whose success depends on the involvement of seven contributors $A, \ldots, G$. Feature a is true if person A contributed to the task, feature b is true if person B contributed and so on. The label y indicates if the task is successfully solved or not. The following table shows two positive and two negative examples for this classification problem.*

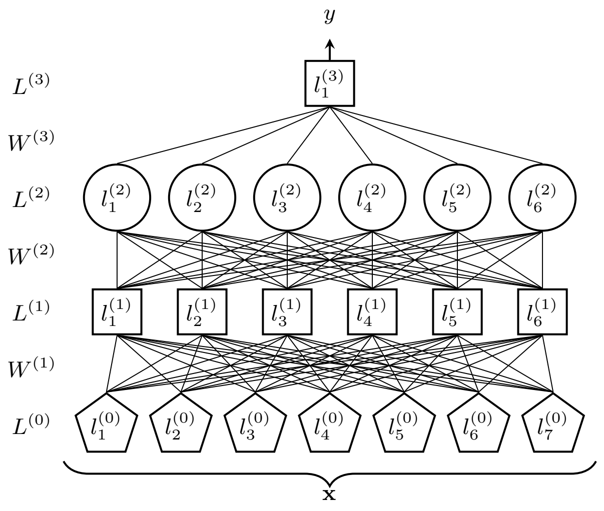
Example	a	b	c	d	e	f	g	y
$\mathbf{x}_1$	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
$\mathbf{x}_2$	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*
$\mathbf{x}_3$	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*
$\mathbf{x}_4$	*1*	*0*	*1*	*0*	*0*	*0*	*1*	*0*

In the classification problem above, we have seven features, hence we get up to $2^7 = 128$ instances. A subset of these are used as training data for the task of training a so-called NNF network (see below), so that the output of the network is consistent with the training data and generalizes on unseen data as well as possible.

In analogy to neural networks, we assume that the network structure is fixed and that only the binary weights have to be optimized. We consider networks with a structure in the style of propositional formulas in NNF resulting in the following definition. An *NNF network* consists of multiple *layers* $\langle L^{(0)}, \ldots, L^{(h+1)}\rangle$ with sizes $d^{(0)}, \ldots d^{(h+1)}$, respectively, where

- $L^{(0)} = (l_1^{(0)}, \ldots, l_d^{(0)}, l_{d+1}^{(0)}, \ldots l_{2d}^{(0)})$ is the $2d$-dimensional input layer consisting of the features and their negations
- h hidden layers $L^{(1)}, \ldots, L^{(h)}$ with varying dimensions $d^{(1)}, \ldots d^{(h)}$;
- $L^{(h+1)}$ is the 1-dimensional output layer, i.e., $d^{(h+1)} = 1$ corresponding to label y

The network encodes a Boolean function over the features and negated features, which are interpreted as Boolean variables. In other words, the input layer receives the features and negated features of an example $\mathbf{x}$ as input, and the output layer is a single node that predicts the binary label y of $\mathbf{x}$.



y
$L^{(3)}$
$l_1^{(3)}$
$W^{(3)}$
$L^{(2)}$
$l_1^{(2)}$ $l_2^{(2)}$ $l_3^{(2)}$ $l_4^{(2)}$ $l_5^{(2)}$ $l_6^{(2)}$
$W^{(2)}$
$L^{(1)}$
$l_1^{(1)}$ $l_2^{(1)}$ $l_3^{(1)}$ $l_4^{(1)}$ $l_5^{(1)}$ $l_6^{(1)}$
$W^{(1)}$
$L^{(0)}$
$l_1^{(0)}$ $l_2^{(0)}$ $l_3^{(0)}$ $l_4^{(0)}$ $l_5^{(0)}$ $l_6^{(0)}$ $l_7^{(0)}$
$\mathbf{x}$

Fig. 1: A fully connected positive NNF network. The edges are binary weights, where $w_{j,k}^{(l)}$ connects node $l_j^{(l-1)}$ and $l_k^{(l)}$. Its input is a binary feature vector $\mathbf{x}$ at the bottom, the activation is propagated forward via all edges with weights $w_{j,k}^{(l)} = 1$ over disjunctive (circles) or conjunctive (squares) nodes to the final output y.

An NNF network requires the layers to alternate between conjunctions and disjunctions. We typically assume that all nodes in odd-numbered layers $L^{(1)}, L^{(3)}, \ldots$ are logical ANDs, whereas the even-numbered layers $L^{(2)}, L^{(4)}, \ldots$ consist of ORs.

Moreover, we use a fully connected network, that is, each node in layer $L^{(l)}$ is connected to all nodes in layer $L^{(l-1)}$. Each connection represents a *binary weight* $w_{j,k}^{(l)}$, encoding whether node $l_j^{(l-1)}$ in the previous layer $l-1$ is relevant for the connected node $l_k^{(l)}$ in layer l. Thus, $W^{(l)} = \left(w_{j,k}^{(l)}\right)$ is a $d^{(l-1)} \times d^{(l)}$-dimensional Boolean matrix. Once suitable weights have been found, the resulting NNF expression can be obtained by removing all connections with weight 0.

Figure 1 shows an example of a (positive) NNF network with $d = 7$ binary features as input, $h = 2$ hidden layers with $m = 6$ nodes each, and a single output node. This structure uses a total of $7 \cdot 6 + 6 \cdot 6 + 6 \cdot 1 = 84$ weights. It could be trained to learn the NNF expression of Example *1*.

IV. Learning NNF Networks with SAT

Given a Boolean data matrix $\mathcal{D} = (\mathbf{x}_{i,j})_{n\times d}$, we present a SAT encoding for the problem of learning an NNF network that returns the desired output y_i for each training example $\mathbf{x}_i$. For this, we use a set of constraints defined over Boolean *activation values* $a_{i,k}^{(l)}$, which specify whether the k-th node in layer l is activated for example $\mathbf{x}_i$. This allows us to model the problem of learning an NNF with hidden layer sizes $d^{(1)}, \ldots, d^{(h)}$ by the following SAT problem:

- Given the labeled data

  - $\mathbf{x}_{i,j}, y_i \qquad (1 \leq i \leq n,\ 1 \leq j \leq d)$
- For every layer l, find Boolean values
  - $w_{j,k}^{(l)} \qquad (1 \leq j \leq d^{(l-1)},\ 1 \leq k \leq d^{(l)})$

such that the following constraints are satisfied:

- *Input Activation:* the activation of an input feature $\mathbf{x}_{i,j}$ has to be propagated to all nodes in the first (conjunctive) layer for which the weight $w_{j,k}^{(1)} = 1$. Hence, we get $w_{j,k}^{(1)} \rightarrow \mathbf{x}_{i,j}$ for every input feature. That is, if the weight $w_{j,k}^{(1)}$ is 1, then $\mathbf{x}_{i,j}$ is activated in $a_{i,k}^{(1)}$, and if the weight $w_{j,k}^{(1)}$ is 0, then the formula collapses just to $\top$. The case of negated input features is analogous. Formally,

$$a_{i,k}^{(1)} = \bigwedge_{j=1}^{d} (w_{j,k}^{(1)} \rightarrow \mathbf{x}_{i,j}) \wedge \bigwedge_{j=d+1}^{2d} (w_{j,k}^{(1)} \rightarrow \neg \mathbf{x}_{i,j-d}) \quad (2)$$

  The second conjunction can be ignored in case of learning positive NNFs.
- *Disjunctive Activation:* for a disjunctive layer l, each node k must be activated for example $\mathbf{x}_i$ if at least one of the nodes j in the conjunctive layer $l-1$ for which the connection is enabled ($w_{j,k}^{(l)} = 1$) is activated. Formally,

$$a_{i,k}^{(l)} = \bigvee_{j=1}^{d^{(l-1)}} (w_{j,k}^{(l)} \wedge a_{i,j}^{(l-1)}) \quad (3)$$

  if l is even $(2 \leq l \leq h)$.
- *Conjunctive Activation:* for a conjunctive layer, each node k must be activated for example $\mathbf{x}_i$ if all nodes j in the disjunctive layer $l-1$ for which the connection is enabled ($w_{j,k}^{(l)} = 1$) are activated. Formally,

$$a_{i,k}^{(l)} = \bigwedge_{j=1}^{d^{(l-1)}} (w_{j,k}^{(l)} \rightarrow a_{i,j}^{(l-1)}) \quad (4)$$

  if l is odd $(2 \leq l \leq h)$.
- *Output Activation:* the output activation of the network for example $\mathbf{x}_i$ must correspond to its label y_i in the training data. Depending on whether the output node is conjunctive or disjunctive, this results in

$$\begin{aligned} y_i &= \bigwedge_{j=1}^{d^{(h)}} (w_j^{(h+1)} \rightarrow a_{i,j}^{(h)}) \quad \text{if } h+1 \text{ is odd, or} \\ y_i &= \bigvee_{j=1}^{d^{(h)}} (w_j^{(h+1)} \wedge a_{i,j}^{(h)}) \quad \text{if } h+1 \text{ is even.} \end{aligned} \quad (5)$$

Example 3. *To learn a multi-layered positive NNF of our running example using the structure of Figure 1, the following propositional formula is generated:*

1) *We show the encoding of the input activation without negation for the four examples of the table in Example 2. For the first layer, we assume size six ($1 \leq k \leq 6$). Constant values of the features are already eliminated, e.g., $(0 \vee \neg w)$ simplifies to $\neg w$.*

$$\begin{aligned} a_{1,k}^{(1)} &= 1 && \textit{example } \mathbf{x}_1 \\ a_{2,k}^{(1)} &= (\neg w_{2,k}^{(1)} \wedge \neg w_{4,k}^{(1)} \wedge \neg w_{5,k}^{(1)}) && \textit{example } \mathbf{x}_2 \\ a_{3,k}^{(1)} &= (\neg w_{1,k}^{(1)} \wedge \ldots \wedge \neg w_{5,k}^{(1)} \wedge \neg w_{7,k}^{(1)}) && \ldots \\ a_{4,k}^{(1)} &= (\neg w_{2,k}^{(1)} \wedge \neg w_{4,k}^{(1)} \wedge \neg w_{5,k}^{(1)} \wedge \neg w_{6,k}^{(1)}) \end{aligned}$$

2) *For a disjunctive layer of size six ($1 \leq k \leq 6$), we get:*

$$a_{i,k}^{(2)} = \bigvee_{j=1}^{6} (w_{j,k}^{(2)} \wedge a_{i,j}^{(1)})$$

3) *The output activation is conjunctive. For $i \in \{1,2\}$ (examples 1,2) and $m \in \{3,4\}$ (examples 3, 4) we get*

$$1 = \bigwedge_{j=1}^{6} (w_j^{(3)} \rightarrow a_{i,j}^{(2)}) \qquad 0 = \bigwedge_{j=1}^{6} (w_j^{(3)} \rightarrow a_{m,j}^{(2)})$$

Solving the SAT problem results in settings for all binary weights $w_{j,k}^{(l)}$ which can be decoded into an NNF that holds for all positive and no negative training examples. Figure 2 shows a possible solution that is automatically found by a SAT solver.

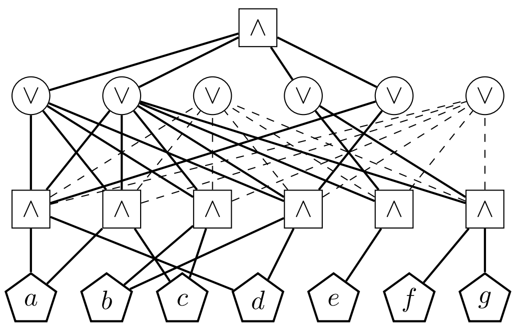


Fig. 2: Example of an NNF networks trained on the complete truth table for the Boolean function $(a \vee b) \wedge (c \vee d) \wedge [e \vee (f \wedge g)]$ by our SAT-based method. We only depict non-zero weights and nodes with at least one non-zero input. Weights of nodes that do not influence the output expression are dashed.

V. Experimental Evaluation

The goal of our experiments is two-fold. Firstly, we want to compare the performance of our exact algorithm to the performance of a greedy algorithm for learning NNF networks. To this end, we follow [9] and evaluate the results for three types of NNFs with varying depths, comparing those learned with our SAT-based approach (SAT) to their stochastic hill-climbing search (SHS):

NNF(1): A shallow network with a single hidden layer of dimension 20, which effectively learns a DNF expression.

NNF(3): A deep network with three hidden layers of dimensions $32, 8, 2$, respectively.

NNF(5): A deep network with five hidden layers of dimensions $32, 16, 8, 4, 2$, respectively.

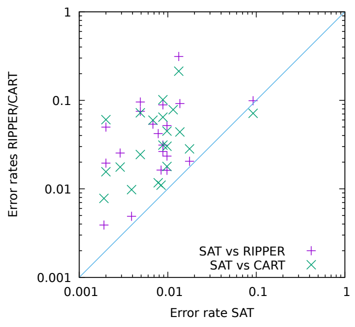
Secondly, we want to compare the performance of our SAT-based approach to the state-of-the-art rule learning algorithm Ripper [13] and the decision tree learner Cart [14]. Since

Fig. 3: Results of networks with depth 1, 3, and 5, optized by SAT or stochastic hill-climbing, as well as the benchmark algorithms RIPPER and CART on the 20 artificial and 7 UCI datasets.

(a) Average Accuracies and Average Ranks

	NNF(5)		NNF(3)		NNF(1)		RIPPER	CART
	SAT	SHS	SAT	SHS	SAT	SHS		
Artificial Datasets								
Ø acc.	**0.9857**	0.9467	**0.9852**	0.9502	***0.9939***	0.9386	0.9591	0.9644
Ø rank	2.7	5.9	2.7	6.05	1.45	7	4.3	4.4
UCI Datasets								
Ø acc.	**0.9682**	0.9229	**0.9722**	0.9190	***0.9807***	0.9153	0.9290	0.9289
Ø rank	4.1429	4.4286	3.1429	5.7143	2.1429	5.8571	5.5714	4.2857

(b) Error Rate Comparison of NNF(1)



these algorithms essentially learn DNFs, we compare them to our SAT-based algorithm on the shallow network NNF(1).

To evaluate our SAT-based approach, we use, on the one hand, the 20 artificially generated datasets from [9], and on the other hand seven datasets from the UCI Repository [15] (*car-evalaution, kr-vs-kp, monk-1, monk-2, mushroom, tic-tac-toe, vote*). We match the experimental setup of [9], so that our results are directly comparable. In particular, we also used a stratified 10-fold cross-validation for estimating accuracies.

All experiments were run with a timeout of 72 hours on a cluster of dual-socket AMD EPYC 7313 @ 3.7GHz machines running Ubuntu 24.04 with a 64 GB memory limit per computation. To solve the resulting SAT problems, we use the SAT solver CADICAL [16], which we interface via the PYSAT toolkit [17].

a) Accuracy: The results are summarized in Figure 3. To the left, Table 3a shows the average accuracies of all tested algorithms over the artificial and UCI datasets. It can be seen that our SAT-based encoding outperforms the stochastic search in terms of average accuracy on both the artificial and UCI datasets. This is also confirmed by the results in terms of average rank, which show the rank (1–8) of each of the eight algorithms averaged over all datasets.

Interestingly, the flat networks NNF(1) outperform the deeper networks, as well as the benchmark algorithms RIPPER and CART. The latter can also be seen from Figure 3b on the right, which shows the logarithmic error rates of RIPPER and CART relative to those of NNF(1) on the artificial datasets. The superior performance of NNF(1) somewhat contradicts the findings of [9], who argued that deeper structures should work better on these datasets. We suspect that this is caused by the fact that our approach finds an exact fit of the model to the training data. This increases the risk of overfitting, in particular for the deeper structures which have a considerably higher number of parameters.

b) Memory and Run-time: On average, for the artificial datasets, performing the full 10-fold cross validation for the flat network NNF(1) required around 550 megabytes of memory and took about 2 minutes, whereas for NNF(5) these numbers increased to 1.7 gigabytes and 17 minutes, respectively. The SAT problems associated with these datasets also grew in complexity with the network depth: for NNF(1), the SAT encoding consisted of around 1.3 million clauses in 440 thousand variables, while for NNF(5) it increased to 4 million clauses in 1.3 million variables. The number of variables that need to be processed by the SAT solver grows as $O(n \cdot m^2 \cdot h)$, where n is the number of examples, m is the maximum number of nodes in a hidden layer, and h is the number of hidden layers. For example, training the NNF(5) network for our largest dataset `mushroom`, which consists of more than 8100 examples with 116 features, required solving a SAT problem with more than 180 million clauses in almost 62 million variables. This makes our method slower than the greedy algorithms we have considered, like RIPPER, CART, and SHS. To overcome this scalability obstacle, we suggest computing *optimal local modifications*, according to the principle of Local Improvement Based on SAT (SLIM). This approach has already been highly successful in decision tree induction and Bayesian network learning [6], [18].

VI. CONCLUSIONS AND FUTURE WORK

We proposed a SAT-based method for learning negation normal form (NNF) networks with Boolean weighted connections directly from input-output data and have connected it to recent work in deep rule learning. In particular, we have shown that an exact solution for the problem — within the bounds of the given network structure — can be obtained by encoding it as a satisfiability problem. Experimental results showed that our exact approach outperforms previous work in deep rule learning that used a stochastic optimization algorithm in terms of accuracy. Moreover, we showed that our SAT-based approach on a flat network outperforms state-of-the-art rule-based learning algorithms RIPPER and CART. Our results indicate that SAT-based learning can serve as a powerful tool in the synthesis of compact, interpretable, and verifiable networks opening many directions for future work.

While the presented approach is promising in terms of accuracy, efficiency needs to be improved. So far, we did not implement any optimizations which we leave to future work. Further, we want to develop approaches beyond NNF and exploit less restricted structures. We also plan to leverage the power of modern incremental SAT solvers for iteratively refining the learned network.

References

[1] E. Angelino, N. Larus-Stone, D. Alabi, M. I. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists for categorical data," *Journal of Machine Learning Research*, vol. 18, pp. 234:1–234:78, 2017. [Online]. Available: http://jmlr.org/papers/v18/17-716.html

[2] V. Q. P. Huynh, J. Fürnkranz, and F. Beck, "Efficient learning of large sets of locally optimal classification rules," *Machine Learning*, vol. 112, no. 2, pp. 571–610, 2023.

[3] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[4] R. Evans, D. Saxton, D. Amos, P. Kohli, and E. Grefenstette, "Can neural networks understand logical entailment," in *Proceedings of the International Conference of Learning Representations (ICLR)*, 2018. [Online]. Available: https://arxiv.org/abs/1802.08535

[5] J. Yu, A. Ignatiev, P. J. Stuckey, and P. L. Bodic, "Learning optimal decision sets and lists with SAT," *Journal of Artificial Intelligence Research*, vol. 72, pp. 1251–1279, 2021.

[6] A. Schidler and S. Szeider, "SAT-based decision tree learning for large data sets," *Journal of Artificial Intelligence Research*, vol. 80, pp. 875–918, 2024.

[7] S. Dash, O. Günlük, and D. Wei, "Boolean decision rules via column generation," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Montréal, Canada, 2018, pp. 4660–4670. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/743394beff4b1282ba735e5e3723ed74-Abstract.html

[8] D. Wei, S. Dash, T. Gao, and O. Günlük, "Generalized linear rule models," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 2019, pp. 6687–6696. [Online]. Available: http://proceedings.mlr.press/v97/wei19a.html

[9] F. Beck and J. Fürnkranz, "An empirical investigation into deep and shallow rule learning," *Frontiers in Artificial Intelligence*, vol. 4, p. 145, 2021.

[10] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190009T

[11] H. Mhaskar, Q. Liao, and T. A. Poggio, "When and why are deep networks better than shallow ones?" in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, S. P. Singh and S. Markovitch, Eds. San Francisco, California, USA: AAAI Press, 2017, pp. 2343–2349. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14849

[12] A. Darwiche and P. Marquis, "A knowledge compilation map," *Journal of Artificial Intelligence Research*, vol. 17, pp. 229–264, 2002.

[13] W. W. Cohen, "Fast effective rule induction," in *ML95*. Morgan Kaufmann, 1995, pp. 115–123.

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

[15] M. Kelly, R. Longjohn, and K. Nottingham. (2025) The UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu

[16] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Gurfinkel and V. Ganesh, Eds., vol. 14681. Springer, 2024, pp. 133–152.

[17] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python Toolkit for Prototyping with SAT Oracles," in *Theory and Applications of Satisfiability Testing – SAT*, 2018, pp. 428–437.

[18] V. P. Ramaswamy and S. Szeider, "Turbocharging treewidth-bounded bayesian network structure learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*. AAAI Press, 2021, pp. 3895–3903.