# RL-Net: Interpretable Rule Learning with Neural Networks

Lucile Dierckx[1,2][0000−0003−2855−1042], Rosana Veroneze[2,3][0000−0003−4007−9350], and Siegfried Nijssen[1,2][0000−0003−2678−1266]

[1] TRAIL Institute, Louvain-la-Neuve, Belgium
[2] ICTEAM/INGI, UCLouvain, Louvain-la-Neuve, Belgium
[3] FEEC/DCA, Unicamp, Campinas-SP, Brazil
{firstname.lastname}@uclouvain.be

**Abstract.** As there is a need for interpretable classification models in many application domains, symbolic, interpretable classification models have been studied for many years in the literature. Rule-based models are an important class of such models. However, most of the common algorithms for learning rule-based models rely on heuristic search strategies developed for specific rule-learning settings. These search strategies are very different from those used in neural forms of machine learning, where gradient-based approaches are used. Attempting to combine neural and symbolic machine learning, recent studies have therefore explored gradient-based rule learning using neural network architectures. These new proposals make it possible to apply approaches for learning neural networks to rule learning. However, these past studies focus on unordered rule sets for classification tasks, while many common rule-learning algorithms learn rule sets with an order. In this work, we propose RL-Net, an approach for learning ordered rule lists based on neural networks. We demonstrate that the performance we obtain on classification tasks is similar to the state-of-the-art algorithms for rule learning in binary and multi-class classification settings. Moreover, we show that our model can easily be adapted to multi-label learning tasks.

**Keywords:** Interpretability · Pattern Set Mining · Rule Learning · Binary Neural Networks

## 1 Introduction

Organizations are increasingly using Machine Learning models to help decision-making. For many application domains (such as medicine, health care, criminal justice, and education), interpretability is essential in addition to predictive performance. Therefore, white-box models are preferable to black-box models in these scenarios.

Rule-based classification models are an important class of interpretable models, which provide symbolic white-box models that are expressed as simple IF-THEN rules. A distinction can be made between rule *sets* and rule *lists*. In a

rule list, the rules have an order, and the first rule of which all conditions in the IF-part are satisfied is used to perform a prediction. An advantage of this approach is that the resulting models are also interpretable on classification tasks with more than two classes: one rule is used to perform the prediction. Models based on rule sets typically rely on voting, where all rules vote for classes, or they only work for two classes. This makes these methods less interpretable. For this reason, we focus on rule lists in this work.

Roughly speaking, two classes of approaches for learning rule-based models can be distinguished. A common strategy for rule learning relies on *pattern mining*. Traditional pattern mining is formulated as the problem of computing $\mathrm{Th}(\mathcal{L}, \varphi, \mathcal{D}) = \{\pi \in \mathcal{L} | \varphi(\pi, \mathcal{D}) \text{ is true}\}$, where $\mathcal{D}$ is the dataset, $\mathcal{L}$ is a language of patterns, and $\varphi$ is a constraint, often based on support [9,10]. The size of the search space of this problem is exponential in the size of $\mathcal{L}$.

The number of all patterns satisfying the constraints is usually too large. Thus, patterns are often post-processed in a step-wise procedure to become useful [10]. In the first step, the patterns that meet the constraints are enumerated. In the second step, some patterns are selected and combined. Again, we have another search space of exponential size, in this case in the size of $\mathrm{Th}(\mathcal{L}, \varphi, \mathcal{D})$.

Most methods adopt heuristics to select and combine the patterns, which is commonly the case for associative classification proposals, such as CBA [12] and CMAR [11]. These methods solve one particular instance of the *pattern set mining* problem, which consists in computing $\mathbf{Th}(\mathcal{L}, \varphi, \psi, \mathcal{D}) = \{\Pi \subseteq \mathrm{Th}(\mathcal{L}, \varphi, \mathcal{D}) | \psi(\Pi, \mathcal{D}) \text{ is true}\}$, where $\psi$ expresses constraints that have to be satisfied by the overall pattern set [9,10]. The major drawback of the step-wise procedure is that it does not scale well.

The second class of methods scales better. Instead of first mining patterns, these approaches learn the rules themselves also using heuristics; typically, they use a heuristic to iteratively add the most promising condition to a rule. A well-known representative of this class is the RIPPER [5] algorithm. While, as a consequence, these approaches find rules more quickly, the heuristics are often specific to one learning task and may have as effect that the algorithm overlooks good rules.

In many application domains of machine learning, recent advances in learning deep neural networks have led neural network techniques to become the state-of-the-art. Search strategies in the neural network literature are very different, and are often based on the use of gradient descent techniques. The success of neural methods has had as effect that many learning problems have now been phrased and solved using these gradient descent techniques. However, traditional neural networks are not interpretable models.

Therefore, the research community has been looking for strategies to combine symbolic forms of Artificial Intelligence with techniques based on neural networks, leading to techniques for neuro-symbolic AI. In the case of rule learning, this could lead to a combination of the interpretability of rule-based models with the search strategies employed when learning neural networks. Recently, Qiao et al. [14] and Fischer & Vreeken [7] developed pattern set mining strate-

gies that rely on binarized neural networks. Adopting a neural network trained with gradient descent methods has several advantages. Indeed, all advances in the area of neural networks have the potential to be leveraged for pattern set mining. This includes stochastic well-developed gradient descent algorithms, well-developed loss functions, well-developed regularization concepts, sophisticated development frameworks, and powerful computing platforms. However, none of these approaches studied how to learn *ordered* rule models for a wide range of learning tasks, including multi-class and multi-label classification tasks, and it is not clear how well neural network-based techniques would work on this task.

In this work, we focus on such problems by extending the Decision Rules Network (DR-Net) proposal of Qiao et al. [14]. We incorporated the possibilities of (a) using hierarchy among the rules, hence adding the possibility of learning classifiers based on *rule lists* in addition to *rule sets*, and (b) solving multi-class classification problems. Furthermore, the consequent part of the rules (i.e., the class labels) is fixed in DR-Net, so all learned rules have the same consequent. We instead learn the class label of each rule together with the condition. Lastly, our proposal can easily be tweaked to solve multi-label classification problems.

This paper is organized as follows: Sect. 2 gives an overview of the related work. Section 3 presents the architecture of our proposed interpretable Rule Learning neural Network (RL-Net). The datasets used, the models for comparison, the experimental protocol, as well as the obtained results, are presented in Sect. 4. Finally, Sect. 5 concludes this paper.

## 2 Related Work

The use of neural networks to learn rule-based classifiers is still in its early stages.

A first example is the work of Beck and Fürnkranz [2,3] which learns rule sets to perform binary classification using a network structure. However, the network weights are learned using a greedy heuristic instead of a differentiable approach.

Yang et al. [16] presented the Deep Neural Decision Tree (DNDT) method that mimics the structure of a decision tree using a neural network architecture. In this proposal, the weights are trained with a gradient descent algorithm. The splitting value for each attribute and the rule labels are learned during the training. The model is thus suitable for binary and multi-class classification. The key limitation of their proposal is that it is not scalable w.r.t the number of features. In their experiments, they could only find an accurate single tree for datasets with at most 12 features. Moreover, the limitation of a tree structure makes it impossible to learn arbitrary rule lists.

The Explainable Neural Rule Learning (ENRL) method [15] also learns rules in a differentiable manner coupled with a neural network structure. The Explainable Condition Module (ECM) is the building block of the method. It comprises a feature, an operator, and a value, learning atomic propositions such as $age \geq 18$. Based on the atomic propositions, ENRL adopts a complete binary tree topology to express multiple rules, and the problem of seeking appropriate rules is transformed into a neural architecture search. ENRL creates an ensemble of trees,

and the final decision is made by a voting mechanism, which makes this method less interpretable. Also, it is limited to binary classification.

The Decision Rules Network (DR-Net) method [14] learns rule sets for binary classification. The model is composed of three layers: the input layer, the Rules layer, and the OR layer. The *Rules layer* learns the rules. The number of neurons in this layer is a user-defined parameter that sets the maximum number of rules. Its regularization term controls the length of the rules. The *OR layer* chooses which rules to use and which to ignore. Its regularization term controls the number of rules that will be used in the classifier. The network training is done in two alternating phases, one for the Rules layer and one for the OR layer. This method does not identify rule lists, but rule sets.

As can be seen, most existing methods focus on binary classification, and none of them can find arbitrary rule lists. In this work, we contribute to a neural network for learning rule-based classifiers that are fully interpretable, and suitable for binary and multi-class classification. Our proposal takes full advantage of the advances in neural network literature.

## 3 Approach

This section introduces the details of our contribution.

### 3.1 RL-Net

Our Rule Learning neural Network (RL-Net) was conceived to learn interpretable rule lists that can perform multi-class classification. It can also be easily tweaked to be used in multi-label experiments. RL-Net employs the structure of a neural network as well as its gradient optimization learning methods.

The network is composed of four layers, as depicted in Fig. 1. The first layer is the input layer that receives the dataset's features. It is connected to the rule layer, where the rule conditions are learned. The next layer expresses the hierarchy among the rules, which is necessary to learn a rule list instead of a rule set. Finally, the output layer assigns a specific class label to each rule.

Each layer is presented in more detail in what follows. The method implementation can be found in our GitHub repository on `https://github.com/luciledierckx/RLNet`.

**Input Layer** We assume that the features that are fed to the network are binary. As discussed in the rule layer description, there is no need to duplicate the input dataset to express the negation of a feature because the network can express that by itself. The number of nodes in this layer is equal to the number of binarized features of the dataset.

**Rule Layer** This layer mimics the behavior of logical ANDs. It is composed of $r$ nodes, where $r$ is a user-defined parameter that specifies the number of rules to be learned. This layer and the input layer are the same as in DR-Net[14], while

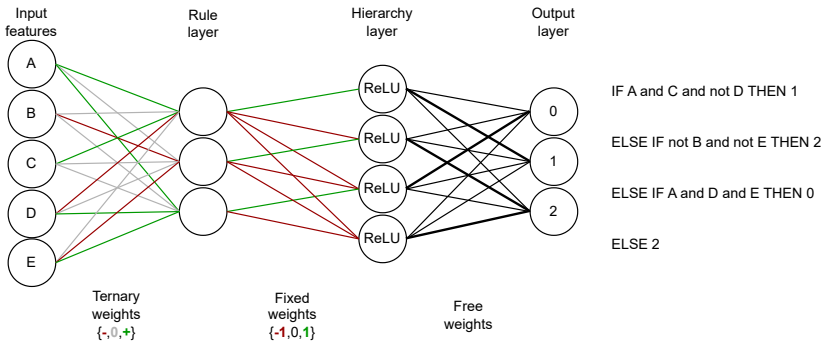



Fig. 1: Global architecture of RL-Net for learning a rule list composed of three rules and the default rule (*else*) for an input dataset consisting of five binary features and three class labels. Green (resp. red) weights represent weights with a positive (resp. negative) value. The edges in bold represent the edges with the highest weight for each node in the hierarchy layer. Connections between neurons that are not represented are non-trainable zero weights.

the next ones are different. Each weight of this layer can either be negative, zero, or positive to represent the fact of using a feature in the rule (+), using the negation of that feature (−), or not considering that feature for the rule (0). As it is not simple to learn discrete weights with a gradient descent algorithm, the ternary weights $W_T$ are obtained by an element-wise product of two other matrices:

$$W_T = W_S \circ W_H \tag{1}$$

The weights in the matrices $W_S$ and $W_H$ are floating-point numbers. The weights in $W_S$ will converge to a positive or a negative value during the training, thus deciding if we use the positive form of a given attribute or its negation. The weights in $W_H$ are referred to as hidden weights. They will decide whether an attribute is used in a rule or is ignored. We ensure that these weights converge to 1 or 0 thanks to the method discussed by Louizos et al. [13] to approximate binary random variables with a Bernoulli distribution. To these hidden weights, a sparsity-based regularization [13] is added to push the weights toward zero and, therefore, obtain shorter rules.

The neurons of the rule layer have to mimic the behavior of a logical AND, that outputs *true* (1) when all rule conditions are met or *false* (0) otherwise. This is done in two steps, as proposed in [14].

In the first step, a neuron from the rule layer performs the following operation:

$$y = \sum w_i \cdot x_i - \sum_{w_i > 0} w_i + 1, \tag{2}$$

where $w_i \in W_T$ and $x_i \in \{0, 1\}$ is the value of the binary feature $i$. In this formulation, the bias has a dynamic value that depends on the number of positive

weights for the neuron. Note that $y = 1$ can only be obtained when all positive weights are related to inputs equal to 1 and all negative weights are related to inputs equal to 0. Thus, the output computed by equation (2) is within $(-\infty, 1]$.

The second step is the binarization of the output computed by (2), which is given by $b(x) = 1$ if $x = 1$ and $b(x) = 0$ otherwise.

We would like to highlight that we attempted to avoid using the two sets of weights, $W_S$ and $W_H$, but the results were much worse, which validates the design of the Rules layer in DR-Net.

**Hierarchy Layer** This layer expresses the hierarchy among the different rules such that a rule $R_k$ can be activated only if all previous rules $R_1, R_2, ..., R_{k-1}$ were not. The structure of this layer was inspired by [1] and adapted to a neural network architecture. The weights of this layer are set in the initialization and are not trainable, as illustrated in Figure 1. The activation of a neuron of this layer is given by a ReLU function. The neuron $k$ in the rule layer represents the rule $R_k$ in the rule list. It is connected to a neuron $l$ of the hierarchy layer by an edge with weight $w_{kl}$, where $w_{kl}$ is 1 when $l = k$, $-1$ when $l < k$, and 0 otherwise.

The last neuron of the hierarchy layer represents the default rule (*else*). It is the only neuron of this layer that uses a bias with a value equal to 1. This ensures that the default rule will be applied when none of the previous rules are applicable. Thus, the number of nodes in this layer is equal to $r + 1$. As the input values of the hierarchy layer are binary (0 or 1), its output will also be binary.

**Output Layer** The last layer of our network learns the label associated with every rule condition. The number of neurons in this layer is equal to the number of class labels in the input dataset. The weights are free, but we add L2-regularization. The activation function is the softmax. As only one rule at a time is active, only the label with the highest activation (i.e., with the highest weight for the active rule) is considered at prediction time. Thus, the learned rules are fully interpretable.

### 3.2 Training and Tuning of RL-Net

Standard neural network techniques are applied in the training of RL-Net. Indeed, we used the Adam optimization algorithm with the cross-entropy loss function. A callback on the validation loss is also applied.

As observed in standard neural networks, the performance of RL-Net for strongly imbalanced datasets can be improved by using a class-balanced version of the loss. Therefore, a balanced version of the loss is used in the first $e_b$ epochs, where $e_b$ is a user-defined parameter. This parameter can be set to zero for datasets for which there is no need for a balanced loss.

Other important parameters of RL-Net refer to the hidden weights $W_H$. These weights are initialized using a normal distribution with mean $\mu$ and variance $\sigma^2$, which directly impacts the probability of using (or not) an attribute in

a rule. Thus, the choice of these parameters influences the length (specificity) of the rules in the initialization, making the search for a good local optimum more or less hard. Other hyper-parameters that must be tuned are the learning rate, the weight of the sparsity-regularization term of the rule layer, as well as the weight of the L2-regularization of the output layer.

### 3.3 RL-Net for Multi-label Classification

The RL-Net architecture was conceived for multi-class classification, but one of its advantages is that it can easily be transformed into a basic multi-label classifier with two minor changes. The first modification is that the activation of the output layer must be the sigmoid activation instead of the softmax. The second one concerns the loss function, which must be designed for multi-label classification, such as binary cross-entropy loss, focal loss, Huber loss, multi-label margin loss, MSE loss, and L1 loss. For our experiments, we choose the binary cross-entropy loss.

## 4 Experiments

Our experiments were designed to answer the following research questions: How does RL-Net compare against its basis, DR-Net, for binary classification? How does RL-Net compare against state-of-the-art rule-based classifiers, RIPPER and CART, for binary and multi-class classification? Are the simple changes in RL-Net for multi-label classification enough to achieve a satisfactory performance?

### 4.1 Datasets

We selected 7 binary and 6 multi-class datasets for our experiments, among which all binary datasets used in the DR-Net paper [14]. We also chose 2 multi-label datasets to perform a first evaluation on multi-label classification. The datasets all come from the UCI Repository [6] except heloc[4], house[5], yeast[6], and scene[6].

From the DR-Net paper [14], we used adult census (`adult`), magic gamma telescope (`magic`), fico heloc (`heloc`), and home price prediction (`house`). To these, we added internet advertisements (`ads`), king-rook vs. king-pawn (`chess`), and mushroom (`mushroom`). For multi-class classification, we chose car evaluation (`car`), nursery (`nursery`), contraceptive method choice (`contraceptivemc`), page blocks classification (`pageblocks`), pen-based recognition of handwritten digits (`pendigits`), and sensorless drive diagnosis (`drive`). We kept the different classes of the multi-class datasets untouched except for nursery where we merged the class "very_recom" and "recommend" as they represented respectively 2.531% and 0.015% of the class distribution. The multi-label experiments were made with the yeast (`yeast`) and scene (`scene`) datasets.

[4] `https://community.fico.com/s/explainable-machine-learning-challenge`

[5] `https://www.openml.org/d/821`

[6] `https://www.uco.es/kdis/mllresources/`

Table 1: Characteristics of the different datasets: The column #Attributes binarized presents the number of attributes after the different preprocessing steps while all the other columns are computed from the unprocessed dataset.

| Binary and multi-class datasets | #Rows | #Attributes | #Attributes binarized | Proportion of each class | | | |
|---|---|---|---|---|---|---|---|
| Adult | 48842 | 14 | 128 | 0.24, 0.76 | | | |
| Magic | 19020 | 10 | 90 | 0.35, 0.65 | | | |
| House | 22784 | 16 | 132 | 0.70, 0.30 | | | |
| Heloc | 10459 | 23 | 147 | 0.48, 0.52 | | | |
| Mushroom | 8124 | 22 | 111 | 0.52, 0.48 | | | |
| Chess | 3196 | 36 | 38 | 0.52, 0.48 | | | |
| Ads | 3279 | 1559 | 1577 | 0.86, 0.14 | | | |
| Nursery | 12960 | 8 | 26 | 0.33, 0.03, 0.33, 0.31 | | | |
| Car | 1728 | 7 | 21 | 0.70, 0.22, 0.04, 0.04 | | | |
| Pageblocks | 5473 | 10 | 88 | 0.90, 0.06, 0.01, 0.02, 0.02 | | | |
| Pendigits | 10992 | 16 | 135 | 10 classes with equal proportions | | | |
| Contraceptivemc | 1473 | 9 | 34 | 0.43, 0.35, 0.23 | | | |
| Drive | 58509 | 48 | 432 | 11 classes with equal proportions | | | |
| Multi-label datasets | #Rows | #Attributes | #Attributes binarized | #Labels | Cardinality | Density | Distinct |
| Yeast | 2417 | 103 | 927 | 14 | 4.237 | 0.303 | 198 |
| Scene | 2407 | 294 | 2646 | 6 | 1.074 | 0.179 | 15 |

### 4.2 Data Preprocessing

Regarding the data used for training, the first step was to remove the data samples for which the percentage of missing values was $\geq 40\%$. Next, we removed the features for which the percentage of missing values was $\geq 40\%$. The remaining missing values were replaced by the most frequent value in the case of categorical attributes, and by the mean value for numerical features. Lastly, we applied the same feature binarization as the one implemented for the DR-Net. This binarization applies one-hot encoding to the categorical attributes, and quantile discretization to the numerical ones followed by ordinal scaling [8].

The main properties of each dataset before and after the preprocessing are presented in Table 1. The preprocessed datasets are the input for all methods used in our experiments. In that way, we can be sure that any observed difference in performance comes from the model itself and not from the data preprocessing.

### 4.3 Algorithms

We compare RL-Net with two state-of-art rule-based classifiers, CART [4] and RIPPER [5], as well as with DR-Net, which is the basis for RL-Net. We used CART from the *scikit-learn* library and RIPPER from *Weka* (*JRip*). We used the authors' implementation of DR-Net.

### 4.4 Protocol

Some of the datasets (namely adult, pendigits, yeast, and scene) have a predefined test set. In this case, it was used as the test dataset. Otherwise, we created a test dataset by selecting 25% of the data samples in a stratified fashion.

For RL-Net and DR-Net, we tuned the hyper-parameters using stratified 10-fold cross-validation. The number of rules ranges from 2 to 20 in our experiments, but we fixed it to 10 in the hyper-parameter tuning for a matter of time. For both algorithms, we tuned the following hyper-parameters: the number of epochs, the learning rate, and the weight of the sparsity-regularization term. The batch size was set to 5% of the dataset size.

For RL-Net, we also tuned the weight of the L2-regularization of the output layer, the number of epochs for the balanced loss $e_b$, and the mean value $\mu$ of the normal distribution used for the initialization of the hidden weights $W_H$.

We compare the algorithms using the same number of rules. Therefore, we do not need to train the OR layer of DR-Net. Accordingly, we set the weight of the OR layer regularization term to zero, and the network training was focused on the Rules layer. For CART, the number of rules is controlled through the user-defined parameter *max_leaf_nodes*. There is no user-defined parameter to control the number of rules in RIPPER's implementation. However, the *minimal weights of instances within a split* parameter influences the number of rules in the classifier. So, we varied this parameter to find the desired number of rules.

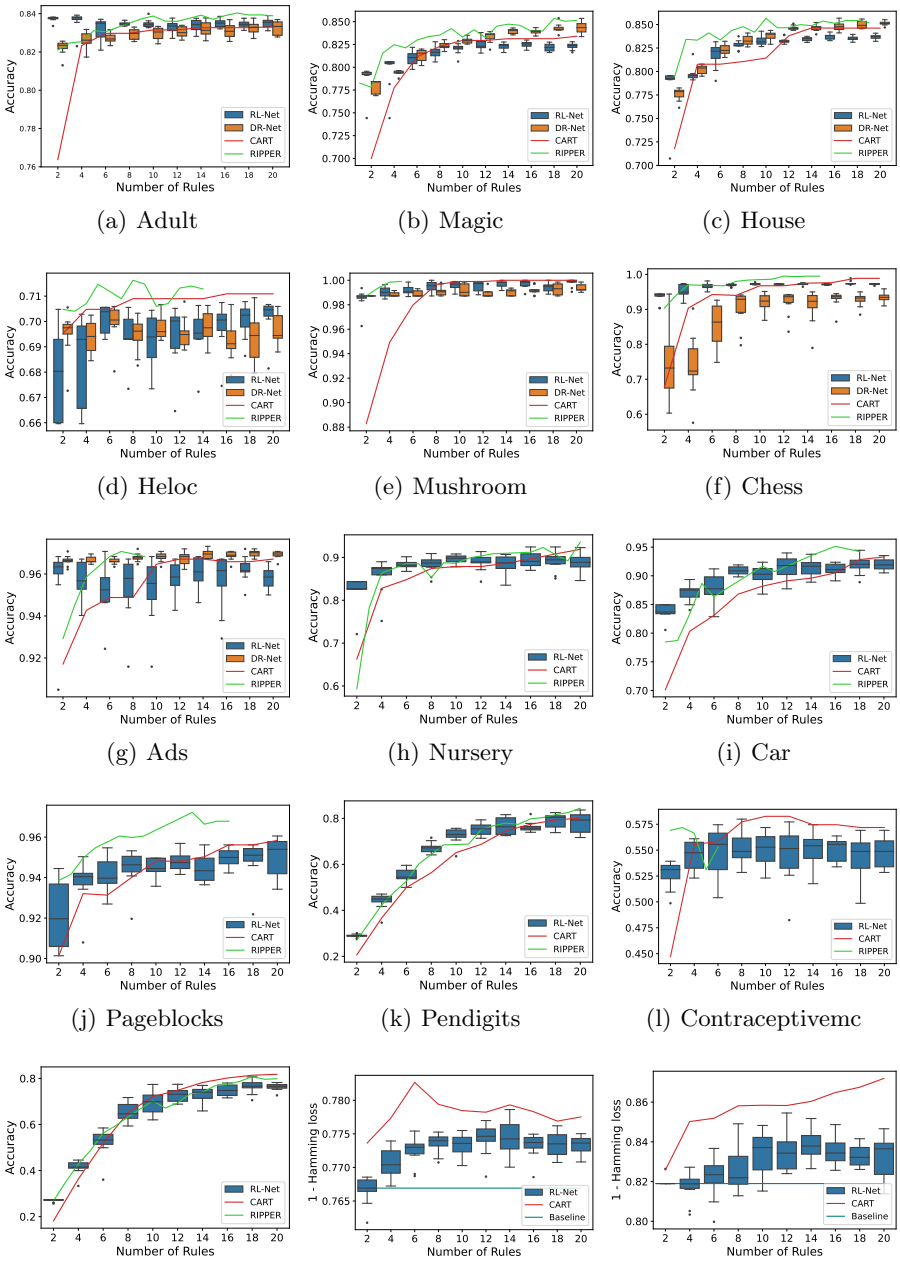
For CART, we also tuned the *criterion* using stratified 10-fold cross-validation.

The remaining hyper-parameters of all algorithms were left to their default values. For ease of reproduction, all details about the hyper-parameter tuning are available in our GitHub repository.

### 4.5 Results

Fig.2 shows the performance of RL-Net and its competitors in terms of accuracy. We set the number of rules from 2 to 20 as our focus is on obtaining interpretable models. RL-Net and DR-Net were run 10 times for each dataset because these methods can get stuck in a poor local optimum depending on the random initialization. The results of these runs are exhibited in a box-plot format.

The binary classification performance is presented in Figs. 2(a)-(g). When we compare RL-Net to DR-Net, we see that it is not possible to say that one of them always performs better than the other. It actually depends on the dataset. For some datasets, such as adult, mushroom, and chess, RL-Net has higher accuracy, for others like ads, it is the other way around. In some other cases, the best-performing method depends on the number of rules considered, such as for magic, house, and heloc. RL-Net has a large performance variability on heloc and ads datasets (with a maximal variability of 10%). In contrast, DR-Net has a large performance variability on the chess dataset (with a maximal variability of 35%). This variability is a drawback of neural networks with such non-standard layers, but it does not stop both networks from achieving competitive performance when the learning does not get stuck in a poor local optimum. RIPPER's performance is generally better than the two neural network approaches, but the difference is not large. RL-Net outperforms CART in a wide range of cases. The performance of the CART decision tree is the most affected when the number of rules is low. Indeed, the length of the rules in the tree is limited by the maximum number of rules, while it is not the case for the other algorithms.




(a) Adult (b) Magic (c) House

(d) Heloc (e) Mushroom (f) Chess

(g) Ads (h) Nursery (i) Car

(j) Pageblocks (k) Pendigits (l) Contraceptivemc

(m) Drive (n) Yeast (o) Scene

Fig. 2: Performance of our RL-Net (blue), the DR-Net (orange), Ripper (green), and CART (red) versus the number of rules, on binary datasets (a to g), multi-class datasets (h to m), and multi-label datasets (n to o).

Figs. 2(h)-(m) present the results for multi-class classification. RL-Net can achieve performance as high as RIPPER for nursery, car, pendigits, and drive datasets. RIPPER outperforms RL-Net for the pageblocks dataset, and when using 2 to 4 rules for the contraceptivemc dataset. Concerning CART, RL-Net can achieve identical performances even though some runs get stuck in poor local optimums as for the binary case. From these results, we can thus clearly see that RL-Net works well on multi-class classification.

The results for multi-label classification are presented in Figs. 2(n)-(o). In addition to CART, we also compare our results with a baseline that, for each class label, predicts the most frequent value (true or false). For both datasets, RL-Net has a lower performance than CART. For the yeast dataset, RL-Net follows CART's performance but scores 1 to 1.5% lower. RL-Net's result for two rules is similar to the baseline one, but RL-Net's performance improves with the number of rules, being up to 1.2% better. For the scene dataset, RL-Net can obtain a lower performance than the baseline when it gets stuck in a poor local minimum, but it clearly outperforms the baseline. However, CART is considerably better than RL-Net for the scene dataset. From these results, we note that using RL-Net for multi-label classification has potential, but its performance is not state-of-the-art yet. This experiment is a proof of concept, indicating that exploring this direction could yield good results.

## 5 Conclusion

Building on the interest in combining neural and symbolic machine learning, in this work we explored gradient-based rule learning using neural network architectures. We implemented and presented our RL-Net method to learn binary and multi-class rule lists using a neural network approach. We showed that with minor adaptations RL-Net can be used to learn multi-label classifiers. We compared our proposal to some other state-of-the-art algorithms for binary and multi-class classification. We also evaluated the potential of RL-Net for learning multi-label tasks. From our results, we concluded that RL-Net is a proper method for learning fully interpretable binary and multi-class classifiers. It does not always achieve the highest performance, but it is never far from the best. Regarding multi-label classification, some additional work should be done to increase RL-Net's performance, but our network architecture is easily compatible with this task, indicating that RL-Net has potential in the integration of rule learning with other neural network-based approaches. Future works include making the method less susceptible to a bad initialization, improving the method for multi-label classification, and integrating RL-Net further with other research in the neural network literature, such as transfer learning, semi-supervised learning, or active learning.

**Acknowledgments** This work was supported by Service Public de Wallonie Recherche under grant n°2010235 – ARIAC by DIGITALWALLONIA4.AI, and under grant n°2110107 - SERENITY2 by WIN2WAL. We would also like to

thank FAPESP, Brazil (Grants No. 2017/21174-8 and 2020/00123-9) for the financial support.

Computational resources have been provided by the supercomputing facilities of the Université Catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

## References

1. Aoga, J.O., Nijssen, S., Schaus, P.: Modeling pattern set mining using boolean circuits. In: International Conference on Principles and Practice of Constraint Programming. pp. 621–638. Springer (2019)
2. Beck, F., Fürnkranz, J.: An empirical investigation into deep and shallow rule learning. Frontiers in Artificial Intelligence **4** (2021)
3. Beck, F., Fürnkranz, J.: An investigation into mini-batch rule learning. arXiv preprint arXiv:2106.10202 (2021)
4. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. Routledge (2017)
5. Cohen, W.W.: Fast effective rule induction. In: Twelfth International Conference on Machine Learning, pp. 115–123. Elsevier (1995)
6. Dua, D., Graff, C.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
7. Fischer, J., Vreeken, J.: Differentiable pattern set mining. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 383–392 (2021)
8. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer Science & Business Media (2012)
9. Guns, T., Nijssen, S., De Raedt, L.: Evaluating pattern set mining strategies in a constraint programming framework. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 382–394. Springer (2011)
10. Guns, T., Nijssen, S., De Raedt, L.: K-pattern set mining under constraints. IEEE Transactions on Knowledge and Data Engineering **25**(2), 402–418 (2011)
11. Li, W., Han, J., Pei, J.: Cmar: Accurate and efficient classification based on multiple class-association rules. In: Proceedings 2001 IEEE international conference on data mining. pp. 369–376. IEEE (2001)
12. Liu, B., Hsu, W., Ma, Y., et al.: Integrating classification and association rule mining. In: Kdd. vol. 98, pp. 80–86 (1998)
13. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through L0 regularization. In: 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings (2018)
14. Qiao, L., Wang, W., Lin, B.: Learning accurate and interpretable decision rule sets from neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 4303–4311 (2021)
15. Shi, S., Xie, Y., Wang, Z., Ding, B., Li, Y., Zhang, M.: Explainable Neural Rule Learning. In: WWW 2022 - Proceedings of the ACM Web Conference 2022. pp. 3031–3041. Association for Computing Machinery, Inc (2022)
16. Yang, Y., Morillo, I.G., Hospedales, T.M.: Deep neural decision trees. ICML Workshop on Human Interpretability in Machine Learning. arXiv preprint arXiv:1806.06988 (2018)