Integrating L0 regularization into Multi-layer Logical Perceptron for Interpretable Classification

Gonzalo Jaimovitch-López[1], Luca Bergamin[2,*], Fabio Aiolli[2] and Roberto Confalonieri[2]

[1]Universidad Internacional Menéndez Pelayo, Santander, Spain
[2]University of Padua, Department of Mathematics, Padua, Italy

Abstract

Deep neural networks are widely used in practical applications of AI, however, their inner structure and complexity made them generally not easily interpretable. Model transparency and interpretability are key requirements in multiple scenarios where not only high performance is enough to adopt the proposed solution. In this work, we adapt a differentiable approximation of L_0 regularization to a logic-based neural network, the Multi-layer Logical Perceptron (MLLP), and we evaluate its effectiveness in reducing the complexity of its interpretable discrete version, the Concept Rule Set (CRS), while preserving its performance. Results are compared to alternative heuristics, such as Random Binarization of the network weights, to assess whether better results can be achieved with a less-noisy technique that sparsifies the network based on the loss function rather than a random distribution.

Keywords

Logical Perceptron, Propositional Network, Interpretable Classification

1. Introduction

Advances in deep learning have promoted the training of models with a continuously increasing number of parameters in the search of higher performance and newer capabilities, reaching the order of billions in some cases. However, some of these solutions became black-box models incapable of explaining the reasoning behind their decisions [1]. Some scenarios with critical use cases such as medicine, law or finance demand for a higher level of explainability. Explainable AI techniques can help in both reducing model complexity while improving the interpretability of the solutions [2].

There are two important aspects of the interpretability of neural networks. The first one, which is a key notion of transparency as explained in [3], is that each part of a model should have an intuitive explanation. In the case of neural networks, the use of activation functions in the neurons makes the understanding of the transformation of each input not feasible for humans. A second aspect is the existing trade-off for interpretability of neural networks among faithfulness, understandability, and model performance. Most of the methods in the literature compromise at least one of those requirements, which might not be suitable for highly sensitive scenarios [4].

Typically, when explaining a black-box model using a surrogate symbolic model (e.g., ruleset, decision tree), accuracy is often sacrificed for transparency. To address this, Wang et al. [5] proposed a hierarchical rule-based model obtained using a Multi-layer Logical Perceptron network (MLLP), where a rule-based model is learned through backpropagation, and discretized later in a ruleset form. A key challenge for rule-based models is finding an easily interpretable, concise structure. In this work, we claim that a sparser network naturally leads to simpler rules. Thus, to achieve higher interpretability, we promote network sparsity through the introduction of a regularization term into the neural network's loss function. Specifically, we apply a differentiable approximation of L_0 regularization [6] and study its effectiveness in aligning the trained continuous model with its discrete, interpretable version.

OVERLAY 24: 6th International Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis
*Corresponding author.
gonzalojaimovitch@gmail.com (G. Jaimovitch-López); bergamin@math.unipd.it (L. Bergamin); fabio.aiolli@unipd.it (F. Aiolli); roberto.confalonieri@unipd.it (R. Confalonieri)
0000-0002-5887-8313 (G. Jaimovitch-López); 0000-0002-0662-7862 (L. Bergamin); 0000-0002-5823-7540 (F. Aiolli); 0000-0003-0936-2123s (R. Confalonieri)

2. Related Work

Rule-based models (often presented as decision trees, rule lists, and rule sets) are widely present in the field due to their transparent inner structure, in contrast to other approaches such as Deep Neural Networks (DNN) which are considered black-box models hard to interpret [7]. Multiple works have explored using backpropagation and/or multilayer structures to learn rule-based models and improve their performance while retaining their transparency [8, 9, 10]. Using differentiable proxies of logic operations is a recently studied topic, due to its easy integration with gradient-based optimization [11].

Model compression for neural networks is a relevant field of study in deep learning since it has been shown that models can be over-parameterized, leading to unnecessary computational resource usage, reduced efficiency, and lower generalization capabilities [2]. Finally, Binary Neural Networks (BNN) are a special type of neural network for which the weights are binary. Courbariaux and Bengio [12] use the straight-through estimator to allow for gradient descent over non-differentiable functions. Even when reducing the computational complexity and improving the transparency, these models are still not considered interpretable due to the complex interactions of non-linear activation functions.

The aim of regularization within the machine learning field is to reduce the generalization error without increasing its training error [13], i.e., to prevent a model from overfitting (which is specially an issue in neural networks due to its generally complex and deep structure), ensuring its performance is aligned for both the training data and new unseen data. Some popular regularization techniques are L_2 norm regularization (also known as weight decay), L_1 norm regularization, or Dropout regularization.

L_0 regularization is a type of regularization that imposes a penalty on the objective function directly by the number of parameters that are non-zero, highly promoting sparsity and improving generalization. Furthermore, it can help speed up inference and training, as those weights that become zero remove some paths in the computational graph. L_0 regularization is an NP-hard problem from the computational complexity perspective [14], which is considered difficult to solve. In [6], an approximate approach to L_0 regularization is proposed, making the problem tractable, with some experimental results that demonstrate its effectiveness in reducing the size of neural networks, such as AlexNet. We argue that the application of this regularization is a great fit for logic-based networks, since sparsity is, in our opinion, a key ingredient to build interpretable classifiers.

3. Background

In this section, we introduce the main notation and concepts used throughout the paper.

3.1. Notation

In the paper, we make use of the following notation. J is a set of binary features and C is a set of class labels. $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$ denotes a training dataset with n instances, where $\mathbf{x_i} \in \{0,1\}^{|J|}$ is a binary feature vector and $\mathbf{y_i} \in \{0,1\}^{|C|}$ is a one-hot class label vector.

$\mathcal{R}^{(l)}$ and $\mathcal{S}^{(l)}$ will denote the l-th layer of the network when l is odd and even, respectively. $\mathcal{S}^{(0)}$ denotes the input layer which consists of $|J|$ input neurons. The output layer consists of $|C|$ output neurons. Given $\mathbf{n}_l$ to denote the number of nodes in the l-th layer, $W^{(l)}$ is a $\mathbf{n}_l \times \mathbf{n}_{l-1}$ matrix containing the weights between the l-th layer and the (l-1)-th layer. Each element of $W^{(l)}$ is referred as $w^{(l)}_{i,j}$.

A rule $\mathbf{r}$ is a conjunction of one or more Boolean variables $\mathbf{r} = b_1 \wedge \ldots \wedge b_k$. A rule set $\mathbf{s}$ is a disjunction of one or more rules $\mathbf{s} = r_1 \vee \ldots \vee r_l$, i.e., $\mathbf{s}$ is a Disjunctive Normal Form (DNF) clause.

Finally, $\sigma(x)$ denotes the sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.

3.2. Concept Rule Set (CRS)

A Concept Rule Set (CRS) is a multi-level hierarchical rule-based model proposed in [5]. A CRS is an instance of a multi-layer logical perceptron network (see Section 3.3) where each weight in the network is binary, i.e., $w^{(l)}_{i,j} \in \{0,1\}$. $w^{(l)}_{i,j} = 1$ indicates that there exists an edge connecting the i-th node in

the l-th layer to the j-th node in the (l-1)-th layer, otherwise $w_{i,j}^{(l)} = 0$. Following the notation adopted in [5], $\mathbf{r}_i^{(l)}$ and $\mathbf{s}_i^{(l)}$ denote the i-th node in layer $\mathcal{R}^{(l)}$ and $\mathcal{S}^{(l)}$, respectively. These nodes are formally defined as follows:

$$\mathbf{r}_i^{(l)} = \bigwedge_{w_{i,j}^{(l)}=1} \mathbf{s}_j^{(l-1)}, \quad \mathbf{s}_i^{(l+1)} = \bigvee_{w_{i,j}^{(l+1)}=1} \mathbf{r}_j^{(l)} \tag{1}$$

Given the above, node $\mathbf{r}_i^{(l)}$ corresponds to a *rule*, while node $\mathbf{s}_i^{(l)}$ corresponds to a *rule set*. In a CRS with L levels, each $\mathcal{R}^{(l)}$ ($l \in \{1, 3, \ldots, 2L-1\}$) is known as a conjunction layer and each $\mathcal{S}^{(l)}$ ($l \in \{2, 4, \ldots, 2L\}$) is known as a disjunction layer. A CRS consists of $2L+1$ layers organized as one input layer followed by pairs of conjunction and disjunction layers.

By providing each input instance as values of the input layer $\mathcal{S}^{(0)}$, once trained, a CRS model works as a classifier $\mathcal{F} : \{0,1\}^{|J|} \to \{0,1\}^{|C|}$. The model outputs the values of nodes in the last disjunction layer $\mathcal{S}^{(2L)}$, where $\mathbf{s}_i^{(2L)} = 1$ indicates that $\mathcal{F}$ classifies the input instance as the i-th class label. The representation learned by the l-th layer in CRS is a binary vector $\mathbf{h}^{(l)}$:

$$\mathbf{h}^{(l)} = \begin{cases} \left[\mathbf{r}_1^{(l)}, \mathbf{r}_2^{(l)}, \ldots, \mathbf{r}_{\mathbf{n}}^{(l)}\right]^\top & l \in \{1, 3, \ldots, 2L-1\} \\ \left[\mathbf{s}_1^{(l)}, \mathbf{s}_2^{(l)}, \ldots, \mathbf{s}_{\mathbf{n}}^{(l)}\right]^\top & l \in \{2, 4, \ldots, 2L\} \end{cases}$$

The value of $\mathbf{h}_i^{(l)}$ is equal to the value of the i-th node in the l-th layer which corresponds to a rule or a rule set. This discrete and explicit representation makes the model transparent.

We refer to the *complexity* of a CRS model as the total length of all rules. We use $|\mathbf{r}_i^{(l)}|$ and $|\mathbf{s}_i^{(l)}|$ to refer to the number of nodes contained in a rule and rule set, respectively. Then, the complexity of a CRS model ($\mathbf{c}_\mathcal{F}$) is defined as follows:

$$\mathbf{c}_\mathcal{F} = \sum_{l=1}^{L} \left(\sum_{i=1}^{\mathbf{n}_l} |\mathbf{r}_i^{(2l-1)}| + \sum_{i=1}^{\mathbf{n}_l} |\mathbf{s}_i^{(2l)}| \right) \tag{2}$$

Wang et al. [5] proposes the use of a MLLP model (introduced in the next section), a neural network architecture, to search for the discrete solution of the CRS model in the continuous space by using gradient descent over the continuous weights of the MLLP model. Subsequently, the weights of the network are binarized to transform the MLLP model into a CRS model, resulting in a classifier which ensures both good performance and transparency. For the discrete CRS extraction, a simple method of binarizing the weights using a threshold is applied.

3.3. Multi-layer Logical Perceptron

The Multi-layer Logical Perceptron (MLLP) is a neural network architecture proposed in [5], designed in such a way that each of its neurons corresponds to one node in the CRS. The main difference with a fully connected Multi-layer Perceptron is the specific design of the activation functions, aiming to replicate the behavior of conjunction and disjunction logical operations. Another important difference is the presence of a selection mechanism that is used to attend to a subset of its inputs.

Given the n-dimensional vectors $\mathbf{h}$ (a layer input vector) and $\hat{W}_i$ (the weights of a given neuron), and $\hat{w}_{i,j} \in [0,1]$ (the weight of the connection between the input h_j and the neuron $\hat{W}_i$), the conjunction and disjunction functions are given by:

$$\text{Conj}(\mathbf{h}, \hat{W}_i) = \prod_{j=1}^{n} F_c(h_j, \hat{w}_{i,j}), \quad F_c(h, w) = 1 - w(1-h), \tag{3}$$

$$\text{Disj}(\mathbf{h}, \hat{W}_i) = 1 - \prod_{j=1}^{n} \left(1 - F_d(h_j, \hat{w}_{i,j})\right), \quad F_d(h, w) = h \cdot w \tag{4}$$

In $\text{Conj}(\mathbf{h}, \hat{W}_i)$, the conjunction operation is obtained by multiplying many values between 0 and 1 together. For $\text{Disj}(\mathbf{h}, \hat{W}_i)$, the disjunction is computed with a similar operation, applying Morgan's law by negating both the inputs and the outputs of the function. To implement this negation, $N(x) = 1 - x$ is used. F_c and F_d act as a selection mechanism. By turning a weight w to zero, F_c can learn to output 1, regardless of its input, leaving the subsequent conjunction operation unaffected. A similar mechanism is implemented for F_d, making sure to output zero instead. Notice that when $\mathbf{h}$ and $\hat{W}_i$ are both binary vectors, Eq. 3 and 4 reduce to the conjunction and disjunction of a subset of the elements in $\mathbf{h}$.

The conjunction and disjunction functions are applied to the neurons in the l-th layer as follows:

$$\hat{\mathbf{r}}_i^{(l)} = \text{Conj}\left(\hat{\mathbf{s}}^{(l-1)}, \hat{W}_i^{(l)}\right), l \in \{1, \ldots, 2L-1\} \quad \hat{\mathbf{s}}_i^{(l)} = \text{Disj}\left(\hat{\mathbf{r}}^{(l-1)}, \hat{W}_i^{(l)}\right), l \in \{2, \ldots, 2L\} \tag{5}$$

where $\hat{\mathbf{r}}_i^{(l)}$ and $\hat{\mathbf{s}}_i^{(l)}$ are neurons in the l-th layer of the MLLP, and $\hat{W}^{(l)}$ is a $\mathbf{n}_l \times \mathbf{n}_{l-1}$ weight matrix.

The weights of the MLLP in the network need to be constrained in the range $[0, 1]$ to ensure the proper functioning of the conjunction and disjunction activation functions. To this end, the weights of a given layer l are constrained using a clip function: $\text{Clip}(\hat{w}_{i,j}^{(l)}) = \max\left(0, \min\left(1, \hat{w}_{i,j}^{(l)}\right)\right)$.
Given the encoding above, the MLLP network can function identically to the corresponding CRS when the weights are set to the same discrete values, while still remaining differentiable.

Let $\hat{\mathcal{F}}$ be the MLLP model and $\hat{W}$ be the weights to be learned by the network. The MLLP loss function is given by:

$$\mathcal{L}(\hat{W}) = \frac{1}{N}\sum_{i=1}^{N} MSE\left(\hat{\mathcal{F}}(\mathrm{x}_i, \hat{W}), \mathrm{y}_i\right) + \lambda \sum_{l=1}^{2L} \left\|\hat{W}^{(l)}\right\|_2^2 \tag{6}$$

The second term in the rhs of Eq. 6 is the L_2 regularization term. This term is included to shrink the MLLP weights and it induces a simpler CRS model.

After training the continuous model (MLLP), a discrete and interpretable model (CRS) can be extracted. Its behavior is not guaranteed to follow the continuous version, and the authors observe a drop in performance. This problem arises with the application of conjunction and disjunction operations over continuous weights, as the MLLP weights can be in the range [0,1]. To mitigate this issue, the authors propose a training method based on Random Binarization (RB).

The RB method selects some of the weights during training using a random binarization rate ($\mathcal{P} \in [0, 1]$), which is the probability of a weight to be binarized. The binarized weights are frozen and the forward (predict) and backpropagation are performed. After a given number of steps, the binarized weights are restored and a new set of random weights is binarized (in the experiments, the RB operation is applied every epoch). Therefore, during training, the MLLP model has a more aligned behaviour to that of the CRS, what ensures a closer performance between the MLLP and CRS models after the binarization conversion of the continuous solution. $\tilde{W}_i^{(l)}$ denotes the weights of the i-th neuron in the l-th layer of the MLLP after RB. These weights replace $\hat{W}_i^{(l)}$ in Eq. 5. The behavior of the RB method is similar to that of the Dropout regularization [15], which helps addressing overfitting.

3.4. L_0 regularization Approximation

In this work, we focus on the L_0 regularization approximation proposed in [6]. The authors propose a set of gates that determine whether a parameter or group of parameters of the network (e.g., all the weights associated to a given input neuron of a layer of the network) are active (i.e., value greater than 0) or inactive (i.e., value equal to 0). The values z of those gates can be drawn from a distribution such as the binary Bernoulli. However, this distribution needs to be smoothed in order to be differentiable. Given s as a continuous random variable with a distribution $q(s)$ and parameters ϕ, rectified to be in the interval [0,1], the probability of the gate being active can be calculated using its cumulative distribution function (CDF) so that:

$$s \sim q(s|\phi), \quad z = \min(1, \max(0, s)), \quad q(z \neq 0|\phi) = 1 - Q(s \leq 0|\phi) = Q(s > 0|\phi) \tag{7}$$

The selected distribution q proposed in this work is the *hard concrete* distribution, obtained by stretching a *binary concrete* distribution [16], closely tied to the Bernoulli distribution, and rectifying each sampled value z using a hard-sigmoid to constrain its values in the range $[0, 1]$. The parameters ϕ for this distribution are $\log \alpha$ as the *location* and β as the *temperature* (to control the degree of recovery of the original Bernoulli distribution). ζ and γ determine the stretching of the original distribution. The *hard concrete* distribution can be formalized as follows:

$$u \sim U(0,1),\ s = \sigma\left(\frac{\log u - \log(1-u) + \log \alpha}{\beta}\right),\ \bar{s} = s(\zeta - \gamma) + \gamma,\ z = \min\left(1, \max(0, \bar{s})\right) \quad (8)$$

The nature of the *binary concrete* distribution allows for the reparametrization trick [17], which prevents the introduction of randomness in the gradient descent process when involving sampling from the learnt distribution (as expressed in Eq. 8). On another note, the authors propose *group sparsity* instead of *parameter sparsity* as a means to achieve network speedups. In their approach, the *group sparsity* is presented as input neuron sparsity in the case of fully connected layers.

$$\hat{\mathcal{R}}(\hat{W}, \phi) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}\big(h(\mathrm{x}_i, \hat{W} \times \mathrm{z}), \mathrm{y}_i\big)}_{\mathcal{L}_E(\hat{W})\text{ (Empirical risk)}} + \underbrace{\lambda \sum_{g=1}^{|G|} |g| \Big(Q(s_g > 0 | \phi_g)\Big)}_{\mathcal{L}_{L0}(\phi)\text{ (Group L0 penalty)}} + \underbrace{\lambda' \sum_{g=1}^{|G|} \Big(Q(s_g > 0 | \phi_g) \sum_{j=1}^{|g|} \hat{W}_j^2\Big)}_{\mathcal{L}_{L2}(\phi)\text{ (Group L2 penalty)}} \quad (9)$$

where $|G|$ corresponds to the number of groups and $|g|$ corresponds to the number of parameters of group g. The probability of a gate g being active under the *hard concrete* distribution for the proposed *grouped sparsity* can be expressed as:

$$\Big(1 - Q(s_g \leq 0 | \phi_g)\Big) = \sigma\big(\log \alpha_g - \beta \log \frac{-\gamma}{\zeta}\big) \quad (10)$$

During test, the deterministic values of the gates are obtained with the following estimator:

$$\hat{z} = \min(1, \max(0, \sigma(\log \alpha)(\zeta - \gamma) + \gamma)) \quad (11)$$

4. Integrating L_0 Regularization into MLLP

The original implementation of the MLLP architecture employs the RB method to address the misalignment between the continuous and discrete solutions, while addressing overfitting similarly as Dropout regularization. Although the CRS model learned through the MLLP achieves high transparency and performance, the resulting solutions can still be complex due to the large number of rules involved.

L_0 regularization turns out to be an interesting method to increase the sparsity of a model, which, in turn, directly impacts the interpretability of the CRS model. Integrating L_0 has a number of advantages:

- **Direct Sparsity Induction:** L_0 regularization directly promotes sparse solutions. This can act also as a feature selection process: those input neurons that are not relevant can be ignored. Dropout regularization mainly improves robustness, and solutions are not necessarily sparse.
- **Interpretability:** with the L_0 regularization technique, the result is a sparse model, which means a less complex model. Therefore, the resulting model is inherently more interpretable.
- **Training Efficiency:** the training of sparse models can be made be more efficient, since all the weights that become 0 are no longer part of the computational graph.

For including L_0 regularization into the MLLP architecture, multiple changes need to be considered.

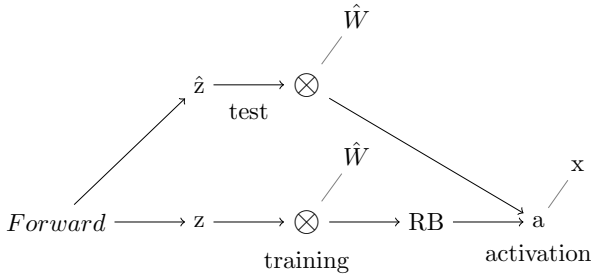

Figure 1: Schema of the forward operation of each conjunction and disjunction layer during training and test for the L_0 regularization adaptation to the MLLP framework.

To start, the MLLP network needs to include the locations $log\ \alpha$ of a *hard concrete distribution* [16] for each gate as trainable parameters. In our implementation[1], we follow the same approach as [6][2], where each gate represents the activation or deactivation of an input neuron, coined as *group sparsity*. These values are initialized by sampling a normal distribution determined by a dropout rate. Furthermore, we define a separate dropout rate for the input layer to the dropout rate used for the rest of the layers.

When performing the continuous *forward* operation, the behavior differs during training and testing. During training, the value z of each gate is sampled from a *hard concrete* distribution, using the reparameterization trick by sampling a uniform distribution. Then, the weights are multiplied with the value of its corresponding L_0 gate, and randomly binarized (when $\mathcal{P} > 0$) before applying the conjunction or disjunction activation functions. During testing, the value of $\hat{z}$ is obtained with a deterministic operation using Eq. 12. We summarize in Fig. 1 the operations of the forward pass.

$$\hat{z} = \min(1, \max(0, \sigma(\log \alpha)(\zeta - \gamma) + \gamma)) \tag{12}$$

A new threshold $\mathcal{T}' = 0.5$ is included to binarize the $\hat{z}$ value of each L_0 gate when applying the (binarized) forward operation to the CRS model. The regularized loss function is defined as follows:

$$\mathcal{L}'(\hat{W}, \phi) = \frac{1}{N}\left(\sum_{i=1}^{N} MSE\left(\hat{\mathcal{F}}(\mathrm{x}_i, \hat{W} \times \mathrm{z}), \mathrm{y}_i\right)\right) + \lambda \mathcal{L}_{L0}(\phi) + \lambda' \mathcal{L}_{L2}(\phi) \tag{13}$$

Finally, since the original implementation of the L_0 regularization is applied to classical neural networks, some minor adaptations were also needed. First, since the weights in the MLLP network are constrained between 0 and 1, the Kaiming initialization strategy outlined in [6] would not make sense. Therefore, the initialization is left in the same way as in [5], using *Uniform*(0, 0.1). Second, MLLP networks do not have any bias term. Therefore, the bias was not considered neither in the neurons nor as locations $log\ \alpha$ of the L_0 gates.

5. Evaluation

In this section, we evaluate the proposed solution following these steps:

1. Analysis of the sparsity achieved using L_0 regularization adapted to the MLLP framework.
2. Comparison between the MLLP baseline and the framework adapted to use the L_0 regularization in terms of CRS and MLLP predictions, as well as complexity of rules in the CRS model.

Dataset. We consider the *connect-4* dataset as our benchmark. This dataset is relevant for rule-learning models [18] due to its considerable size (having 67557 instances), and its difficulty: despite being a deterministic game, i.e., data contains no noise, classifiers struggle to achieve high F1-scores.

[1] https://github.com/gonzalojaimovitch/mllp_l0.git
[2] https://github.com/AMLab-Amsterdam/L0_regularization

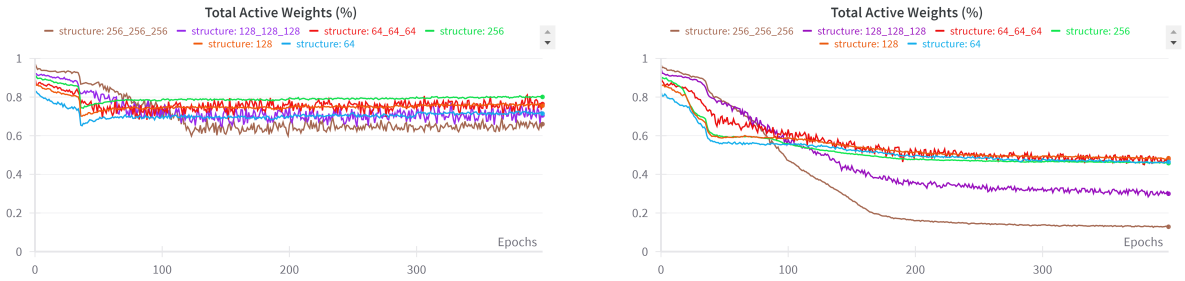

Figure 2: (left) Average percentage of active weights during training of replicated MLLP models for *connect-4* dataset. (right) Average percentage of active weights with L_0 reg. for *connect-4* dataset.

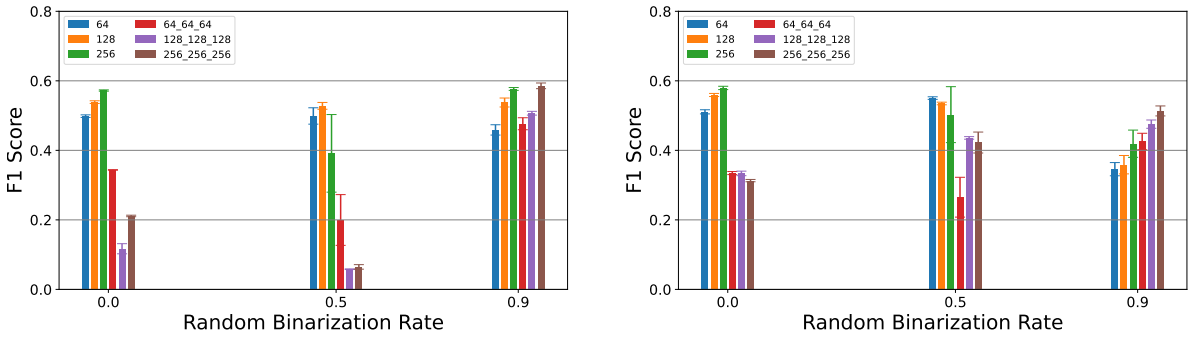

Figure 3: (left) Replicated *connect-4* results for CRS models. (right) *connect-4* results with L_0 reg. for CRS models.

Experimental Settings. The hyperparameter settings are extracted from the Experimental section of [5], namely batch size of 128, 400 epochs, learning rate of 5×10^{-3}, learning rate decay of 0.75 every 100 epochs, weight decay (λ') of 10^{-8}, $\mathcal{T}$ of 0.5, and update of randomly binarized weights every epoch (when applied). Regarding the data, 80% of the training data is used for training, and 20% for validation when performing hyperparameter search. Furthermore, 5-cross validation is adopted for more representative results. For analysing the performance, the F1 score (Macro) is the chosen metric, due to dataset imbalance. The L_0 settings are λ and initial dropout rates are of 0.001, $\mathcal{T}'$ of 0.5, and a value of β fixed to $2/3$ (as recommended by [16]).

Sparsity analysis. Fig. 2 shows the evolution of active weights of the MLLP models during training on the *connect-4* dataset, for both the replicated results and the results including L_0 regularization. An active weight is a weight w which has a value greater than 0. In the case of the models including L_0 regularization, the active weights are those for which the product of the value of the weight w with its corresponding L_0 gate z is greater than 0. As expected, the models including L_0 regularization yield solutions that are sparser than those obtained by the replicated MLLP models. Furthermore, the models with a greater number of weights are sharply sparsed compared to simpler models in terms of active weights. This is more evident in Fig. 2 (right), where it is shown how the active weights are drastically reduced in the case of the model with the largest number of weights: the MLLP model with 3 hidden layers of 256 nodes each.

MLLP performance comparison. Fig. 3 shows the average F1 scores and standard errors for both the replicated results from [5] and the results for CRS models for which L_0 regularization was included. In the case of MLLP models, similar conclusions can be drawn when compared with CRS results. Results are similar or considerably improved with the inclusion of L_0 regularization for both $\mathcal{P} = 0$ and $\mathcal{P} = 0.5$, with special emphasis on the deep models with $\mathcal{P}$ = 0.5. However, when increasing $\mathcal{P}$ to 0.9,

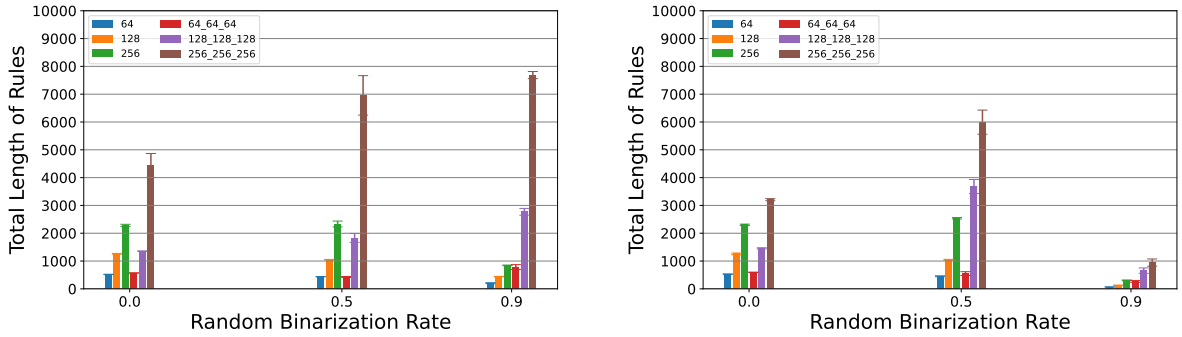

Figure 4: (left) Total length of rules of replicated CRS models for *connect-4* dataset. (right) Total length of rules of CRS models with L_0 for *connect-4* dataset.

scores without L_0 regularization are generally higher, with some exceptions for deeper models. In both cases, the greater the value of $\mathcal{P}$, the worst the performance of the MLLP models, with a similar trend of deterioration.

Rule complexity. Fig. 4 shows the measure of complexity (Eq. 2), as the total length of rules, of the different CRS models for both the replicated results from [5] and the results of CRS models for which L_0 regularization was included during the training of their respective MLLP models. When RB is not applied ($\mathcal{P} = 0$), the total number of rules is pretty similar in both cases, with the exception of the largest model, whose complexity is considerably reduced. In the case of $\mathcal{P} = 0.5$, the complexity of the model with 3 hidden layers of 128 nodes each is increased with the inclusion of L_0 regularization, which might be related to the drastic increase in performance it promotes. When focusing on the largest model, there is a great increase in performance with even a considerable reduction in complexity. Lastly, for $\mathcal{P} = 0.9$, L_0 regularization helps reducing the complexity of almost every model, with special emphasis on the largest model, for which the length of rules is reduced in approximately eight times. This considerable reduction might be related to the slight decrease in performance compared to those models without L_0 regularization.

6. Conclusions and Future Works

In this work, we proposed an adaptation of a computationally complex regularization technique into the MLLP framework, a logical network that acts as the continuous (differentiable) version of a multi-level hierarchical rule-based model. We enhanced the interpretability of a rule-based model by reducing its complexity through a model compression technique, in the form of a regularizer exploited during the optimization. We showed that L_0 regularization can effectively reduce model complexity without affecting its performance, and, in some cases, even enhances it.

As future work, we plan to extend the datasets considered and to introduce the capability of including logical constraints in the network. Specification of data requirements through explicit background knowledge could potentially help the network to meet desirable properties such as safety, consistency and fairness.

References

[1] S. Ali, T. Abuhmed, S. El-Sappagh, K. Muhammad, J. M. Alonso-Moral, R. Confalonieri, R. Guidotti, J. Del Ser, N. Díaz-Rodríguez, F. Herrera, Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence, Information Fusion 99 (2023) 101805. URL: https://www.sciencedirect.com/science/article/pii/S1566253523001148. doi:https://doi.org/10.1016/j.inffus.2023.101805.

[2] M. Sabih, F. Hannig, J. Teich, Utilizing explainable ai for quantization and pruning of deep neural networks, arXiv preprint arXiv:2008.09072 (2020).

[3] Z. C. Lipton, The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery., Queue 16 (2018) 31–57.

[4] V. Swamy, S. Montariol, J. Blackwell, J. A. Frej, M. Jaggi, T. Käser, Interpretcc: Intrinsic user-centric interpretability through global mixture of experts (2024).

[5] Z. Wang, W. Zhang, N. Liu, J. Wang, Transparent classification with multilayer logical perceptrons and random binarization, in: The 34th AAAI Conference on Artificial Intelligence, AAAI, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 6331–6339. URL: https://doi.org/10.1609/aaai.v34i04.6102. doi:10.1609/AAAI.V34I04.6102.

[6] C. Louizos, M. Welling, D. P. Kingma, Learning sparse neural networks through l_0 regularization, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018. URL: https://openreview.net/forum?id=H1Y8hhg0b.

[7] R. Confalonieri, L. Coba, B. Wagner, T. R. Besold, A historical perspective of explainable artificial intelligence, WIREs Data Mining Knowl. Discov. 11 (2021). URL: https://doi.org/10.1002/widm.1391. doi:10.1002/WIDM.1391.

[8] Z. Wang, W. Zhang, N. Liu, J. Wang, Scalable rule-based representation learning for interpretable classification, in: Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS '21, Curran Associates Inc., Red Hook, NY, USA, 2021.

[9] F. Beck, J. Fürnkranz, An empirical investigation into deep and shallow rule learning, 2021. URL: https://arxiv.org/abs/2106.10254. arXiv:2106.10254.

[10] L. Dierckx, R. Veroneze, S. Nijssen, RL-Net: Interpretable Rule Learning with Neural Networks, in: Advances in Knowledge Discovery and Data Mining: 27th Pacific-Asia Conference, PAKDD 2023, May 25-28, Proceedings, 2023. URL: https://dial.uclouvain.be/pr/boreal/object/boreal:274378.

[11] E. van Krieken, E. Acar, F. van Harmelen, Analyzing differentiable fuzzy logic operators, Artificial Intelligence 302 (2022) 103602. URL: http://dx.doi.org/10.1016/j.artint.2021.103602. doi:10.1016/j.artint.2021.103602.

[12] M. Courbariaux, Y. Bengio, Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1, CoRR abs/1602.02830 (2016). URL: http://arxiv.org/abs/1602.02830. arXiv:1602.02830.

[13] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.

[14] T. T. Nguyen, C. Soussen, J. Idier, E.-H. Djermoune, NP-hardness of $\ell 0$ minimization problems: revision and extension to the non-negative setting, in: 13th International Conference on Sampling Theory and Applications, SampTA 2019, 13th International Conference on Sampling Theory and Applications, SampTA 2019, Bordeaux, France, 2019. URL: https://hal.science/hal-02112180. doi:10.1109/sampta45681.2019.9030937.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[16] C. J. Maddison, A. Mnih, Y. W. Teh, The concrete distribution: A continuous relaxation of discrete random variables, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017. URL: https://openreview.net/forum?id=S1jE5L5gl.

[17] D. P. Kingma, M. Welling, Auto-encoding variational bayes, in: Y. Bengio, Y. LeCun (Eds.), 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014. URL: http://arxiv.org/abs/1312.6114.

[18] L. Bergamin, M. Polato, F. Aiolli, Improving rule-based classifiers by bayes point aggregation, Neurocomputing 613 (2025) 128699. URL: https://www.sciencedirect.com/science/article/pii/S092523122401470X. doi:https://doi.org/10.1016/j.neucom.2024.128699.