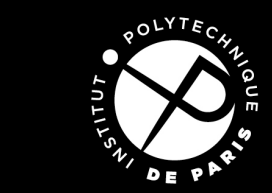
Expressive classification rule learning with an emphasis on learning from sequential data

Marine Collery



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAX091

Thèse de doctorat

Expressive classification rule learning with an emphasis on learning from sequential data.

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Inria Saclay - IBM France Lab - l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Mathématiques et informatique

Thèse présentée et soutenue à Palaiseau, le 12 octobre 2023, par

MARINE COLLERY

Composition du Jury :

Florence d'Alché-Buc Professeur, Telecom Paris	Présidente
Johannes Fürnkranz Professor, Johannes Kepler University Linz	Rapporteur
Céline Hudelot Professeur, CentraleSupélec	Rapporteuse
Claire Glanois Researcher, IT University of Copenhagen	Examinatrice
Siegfried Nijssen Professor, Université Catholique de Louvain	Examinateur
Marc Schoenauer Directeur de Recherche, Inria Saclay	Examinateur
François Fages Directeur de Recherche, Inria Saclay	Directeur de thèse
Shubham Gupta Research Scientist, IBM Research France	Co-encadrant de thèse
Philippe Bonnard Software Engineer, IBM France Lab	Invité
Remy Kusters Research Scientist, Gourmey	Invité

Expressive classification rule learning with an emphasis on learning from sequential data.

Marine Collery

Abstract

During the last decades, machine learning and in particular neural networks have made tremendous progress on classification tasks for a variety of fields such as healthcare, fraud detection or image recognition. They are able to learn from various data types ranging from images to time series and achieve impressive classification accuracy. However, their decisions are difficult or impossible to understand by a human. Rule-based methods on the other end, are interpretable, human-readable and have been widely adopted in different industrial fields with Business Rule Management Systems (BRMS). In practice however, those rules are manually written by experts. One of the reasons manually-written rule models cannot easily be replaced with learned rule models is that rule-based learning models are not able to learn as expressive rules with higher-level concepts and complex grammar. Moreover, due to the lack of latent representations, rule-based learning methods underperform w.r.t. state-of-the-art neural networks.

In this thesis, we propose an end-to-end neural-based approach to learn expressive rules for classification problems. Different levels of expressiveness in rules are presented, implemented and evaluated on some existing datasets and new synthetic ones proposed as new benchmarks for binary classification rule learning. First, the learning of basic disjunctive normal form with a neural network (base model) is studied. Second, extensions to support sequential data are introduced with a recursive and a convolutional approaches. Finally, the model is extended to learn more expressive rules with predefined aggregation functions and overall complex grammar rules.

Résumé

Au cours des dernières décennies, l'apprentissage automatique, et en particulier avec les réseaux de neurones, a fait d'énormes progrès pour résoudre des problèmes de classification dans différents domaines tels que la santé, la détection des fraudes ou la reconnaissance d'images. Ces modèles sont capables d'apprendre à partir de différents types de données, allant des images aux séries temporelles, et d'atteindre une précision de classification impressionnante. Cependant, leurs décisions sont difficiles, voire impossibles à comprendre par un être humain. Les méthodes basées sur des règles, quant à elles, sont interprétables, lisibles par l'homme et ont été largement adoptées dans différents domaines industriels avec les Business Rule Management Systems (BRMS) ou systèmes de gestion des règles métier. En pratique, cependant, ces règles sont écrites manuellement par des experts. L'une des raisons pour laquelle les règles écrites manuellement ne peuvent pas être facilement remplacées par des modèles de règles apprises à partir de données, est que les modèles d'apprentissage de règles ne sont pas capables d'apprendre des règles aussi expressives, avec des concepts de haut niveau et une grammaire complexe. De plus, en raison d'un manque de représentations latentes, les méthodes d'apprentissage basées sur des règles sont moins performantes que les réseaux neuronaux de l'état de l'art.

Dans cette thèse, nous proposons une approche de bout en bout basée sur un réseau de neurones permettant d'apprendre des règles expressives pour des problèmes de classification. Différents niveaux d'expressivité des règles sont présentés et évalués sur de nouvelles données synthétiques et sur certains ensembles de données existants.

Tout d'abord, l'apprentissage d'expressions sous la forme normale disjonctive et sous la forme normale algébrique avec un réseau neuronal (modèle de base) sont étudiées et comparées. Ces modèles sont des réseaux de neurones composés de deux couches, chacune représentant un opérateur logique à l'aide de poids binaire (ET et OU pour la forme normale disjonctive et ET et OU exclusif pour la forme normale algébrique). Il en résulte que la méthode d'apprentissage de règles sous la forme normale algébrique est moins stable

sans apporter de gains de performance. Des pistes pour adapter le modèle de base à différents types de données d'entrées tels que les données numériques ou catégoriques sont ensuite proposées pour étendre le domaine d'application du modèle. De même, pour la sortie du modèle, des directions possibles sont présentées pour supporter des problèmes de multi-classification ou multi-label classification.

Ensuite, des extensions pour prendre en charge les données séquentielles sont introduites avec une approche récursive et une approche convolutive. La première approche repose sur l'ajout d'une récursion sur la première couche du modèle de base à la façon d'un Recurrent Neural Network (RNN). L'architecture de ce genre de modèle conduit a des problèmes d'apprentissage connus qui sont mis en avant dans les résultats. La seconde approche consiste à utiliser le modèle de base comme un filtre dans une architecture convolutive interpretable. Cette approche permet de découvrir des motifs séquentiels de différentes natures en même temps que les règles de classification. Des contraintes dynamiques sont appliquées pendant l'apprentissage pour optimiser la complexité et la qualité des règles.

Enfin, le modèle est étendu pour apprendre des règles plus expressives avec des fonctions d'agrégation prédéfinies et des règles de grammaire complexes. Pour ce faire, le modèle proposé est plus profond et est construit en connectant différents blocs neuronaux d'apprentissage de règles présentés précédemment. La grammaire des règles pouvant être représentées, se rapproche de celle disponible aux utilisateurs de BRMS. Dans un premier temps, une solution pour apprendre des règles avec des agrégats temporels prédéfinis est présentée. Cette solution est ensuite étendue avec un support de plusieurs filtres et de plusieurs types de motifs séquentiels. Ces changements impliquent une augmentation des dimensions du modèle qui conduisent à des difficultés d'apprentissage. Celles-ci sont traitées via de nouvelles propositions d'implémentation pour les couches neuronales du modèle de base. À notre connaissance, il n'existe aucun modèle capable d'apprendre des règles de classification aussi expressives dans la littérature.

Acknowledgments

I would like to express my deepest gratitude to my supervisors François Fages, Philippe Bonnard, Shubham Gupta and Remy Kusters for their guidance and constructive feedback during this PhD project.

François, thank you very much for your academic supervision, your encouragement, your reactivity in all circumstances and for accepting and then supporting the direction the project took after the first year with a neural approach.

Philippe, thank you so much for all these years we have worked together, from the very first project proposal draft to this manuscript. Our discussions, each one more interesting than the other, were crucial to help me move forward, take a step back and get a glimpse of the range of possible directions and applications that we could look into.

Shubham, words cannot express my gratitude to you. Your role in this project was invaluable, from the pertinent technical discussions to the effective brainstorming sessions and your precious feedback on my written and oral contributions. It was a pleasure to work with you and I was very lucky to have you join the project. Thank you !

Remy, you joined the project when I needed it the most, with a challenging second year of PhD, a subject that was just getting clearer and when time started to speed up. It was a pleasure to work on this project together, and our discussions greatly contributed to the project progress.

I would like to express my gratitude to Johannes Fürnkranz and Céline Hudelot for agreeing to review my thesis manuscript and for their precious comments. I also would like to thank the members of the jury Florence D'Alché-Buc, Claire Glanois, Siegfried Nijssen and Marc Schoenauer for evaluating my work and the interesting discussion that followed the presentation. I am also grateful for Benjamin Doerr who made sure that the thesis was going smoothly on all fronts.

I would like to extend my sincere thanks to all the people at IBM who made this project possible: Eric, Stephane, Asma, Nicolas, and Christian of course but also Hagen, Greger and Pierre for their role in the rule learning

incentive at IBM. Special thanks to Stephane for his support over the past 3 years. My colleagues from the France Lab research team, Maxence, Yusik and Laura who I am sure will finish her thesis with flying colors. My colleagues from the Engine team, Benoit, Catherine, Eric, Hugues, Jean Louis and Stephane L.. And of course a special thank you to my colleagues and friends Françoise, Shilpa and Julie for the tea breaks, fun and laughter that were key to put every step of the PhD rollercoaster ride into perspective.

I also would like to thank my colleagues at Inria and in particular members of the Lifeware team: Sylvain, Mathieu, former and current PhD students, Julien, Jeremy, Elea, Sahar and Alexandre, but also Hugo, Guillaume, Henri, Aurélien and Mélanie.

Finally, I could not have undertaken this journey without my friends, family and their invaluable support. Thanks to Suzy, Serge, les inversens, la team STK, le britannia and many others. Above all, thanks to my parents and my brother for their unconditional support, and of course for all the patience, love and immeasurable daily support, *to Mathurin*.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence.
ANF	Algebraic Normal Form.
ARM	Association Rule Mining.
BAL	Business Action Language.
BCRD	Binary Classification Rule Dataset.
BNN	Binary Neural Network.
BOM	Business Object Model.
BRMS	Business Rule Management Systems.
CNN	Convolutional Neural Network.
CR2N	Convolutional Rule Neural Network.
CR2NA	Convolutional Rule Neural Network with Aggregates.
CR2NA-lse	CR2NA with LSE implementation.
CR2NA-soft	CR2NA with Softmax implementation.
DNF	Disjunctive Normal Form.
HMM	Hidden Markov Model.
LLM	Large Language Model.
LSE	LogSumExp.
LSTM	Long-Short Term Memory.
LTL	Linear Temporal Logic.
MDL	Minimum Description Length.
NLP	Natural Language Processing.
ODM	Operational Decision Manager.
RN	Base Rule Network.
RN-anf	ANF Rule Network.
RNN	Recurrent Neural Network.
RR2N	Recurrent Rule Neural Network.
s-CR2NA	Simple CR2NA.
XAI	Explainable AI.
XOM	Execution Object Model.

Notation

$\wedge$	Logical AND operator.
$\vee$	Logical OR operator.
$\neg$	Logical NOT operator.
$\veebar$	Logical XOR operator.
a	A scalar (integer or real).
$\boldsymbol{a}$	A vector.
$\boldsymbol{A}$	A matrix.
$\boldsymbol{A}^\intercal$	Transpose of matrix $\boldsymbol{A}$.
$\mathbf{1}$	A column vector of ones whose dimension can be inferred from the context.
a	A scalar random variable.
$\mathbf{a}$	A vector-valued random variable.
$\mathbf{A}$	A matrix-valued random variable.
$\mathbb{R}$	The set of real numbers.
$\{0, 1\}$	The set containing 0 and 1.
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n.
$[a, b]$	The real interval including a and b.
a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1.
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$.
$\mathrm{a} \sim P$	Random variable a has distribution P.
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1 + \exp(-x)}$.
$\log x$	Natural logarithm of x.
$\odot$	Hadamard product.
$\operatorname{diag}(\boldsymbol{A})$	Main diagonal of a matrix $\boldsymbol{A}$.

Chapter 1

Introduction

Contents

1.1 Context

With the growth of machine learning in the past years due to the newly available computational power combined with a growing number of accessible datasets, improving the quality of learned predictive models has been an important research interest. Today, impressive models are learned but can lack transparency, interpretability and understandability characteristics that are required and essential for numerous application fields. Those models, and especially the ones based on neural networks, are commonly referred to as "black boxes". The focus of the research community is progressively shifting towards providing an explanation for decisions a learned model takes as well as building interpretable, understandable and transparent models from scratch.

Rule-based systems (or rule systems) are transparent and interpretable as the behavior of a rule system is governed by explicit rules. They provide a structured and explicit approach to emulate human reasoning and decision-making processes. For a few decades now, rule systems have been widely adopted in different industrial fields. Business Rule Management Systems (BRMS) offer an intuitive, human readable and comprehensible way to define business rules and hide the computational aspect from the business user.

1.1.1 Interpretability

With the growth of high performance non interpretable black-box models, an important question is raised: to what extent a model can be considered trustworthy, especially for high-stakes decision making ? Different terms are commonly used when referring to this problem. We clarify what we mean by them here for further use.

Model interpretability is the ability (of the model) to present in understandable terms to a human [Li et al., 2021, Doshi-Velez and Kim, 2017]. *Model rationale* is how the model takes decisions. *Interpretable models* are models where the model rationale is inherently understandable by a human. *Explanation methods (or approaches)* explain or translate a model rationale.

Recently, explaining black-box models has attracted considerable research interest under the field of Explainable AI (XAI) (or explainable ML). Explanations are provided by another model (post-hoc) to explain the initial black box model. Also through the analysis of a trained model parameters, mechanistical interpretability methods have recently been introduced [Wang et al., 2022, Elhage et al., 2022] and are considered in the following as part of explanation methods as they are post-hoc methods.

However, for reliability reasons, those aposteriori approaches are not the

solution for high stakes decision-making and more interest should be placed on learning models that are interpretable in the first place [Rudin, 2019]. With their explicit and transparent nature, rules are therefore of great interest for high-stakes decision making.

1.1.2 Rules and Business Rules

At the core of a rule system lies a collection of decision rules that consist of conditions and associated actions. The conditions are typically expressed using logical expressions based on predicate attributes, which evaluate the state of the system or the input data. The actions, on the other hand, define the system's response based on the conditions being met.

`if` conditions `then` actions.

A particular type of rule systems, rule-based classifiers (composed of one or more classification rules) assign a value (or a class) from a set of predefined values to the input data on which it is applied.

The language in use in the rules, called rule language, is defined by a rule grammar that can be formally described by a context-free grammar G.

Context-free grammar [Chomsky, 1956] A context free-grammar is a 4-tuple $G = (V_T, V_N, S, R)$ where

- V_T, is a finite set of terminals or terminal elements in the language that form the alphabet of the language L.
- V_N, disjoint from V_T, is a finite set of non terminal elements (variables) that define a sub-language of L. We note $V = V_N \cup V_T$, the vocabulary of the grammar.
- $S \in V_N$, is the start symbol or variable that defines the whole sentence.
- R is a finite set of rules or production rules of the form $A \to w$ with $A \in V_N$ and $w \in V^*$.

There are also 3 different types of terminal elements on the syntax level:
- reserved words, distinguished with the following style : *reserved*
- signs, such as for example $-, *, ...$
- other terminal elements that are defined prior to the grammar, distinguished with the following style: terminal

For example, for a Disjunctive Normal Form (DNF) expression as condition of a binary classification rule, the grammar G is defined by the following values for V_T, V_N, S and R with conjunction and disjunction operations represented by $\wedge$ and $\vee$ respectively.

$$
\begin{array}{rcl}
V_T & = & \left\{\textit{if}, \textit{then class = 1 else class = 0}, \wedge, \vee, \underline{\text{predicate}}\right\} \\
V_N & = & \{\text{rule}, \text{disjunction}, \text{conjunction}\} \\
S & = & \{\text{rule}\} \\
R & = & \left\{\begin{array}{rcl}
\text{rule} & \rightarrow & \textit{if}\ \text{disjunction}\ \textit{then class = 1 else class = 0} \\
\text{disjunction} & \rightarrow & \text{conjunction} \mid \text{conjunction} \vee \text{disjunction} \\
\text{conjunction} & \rightarrow & \underline{\text{predicate}} \mid \underline{\text{predicate}} \wedge \text{conjunction}
\end{array}\right.
\end{array}
$$

In practice, in this manuscript, the production rules of the grammar of the proposed models will be shared in the main document and the full grammars available in Appendix (see Appendix C).

Expressivity A grammar is considered to be more expressive than others, when it supports more possible non-logically equivalent expressions. Expressivity is, thus, the quality of being expressive.

In practice, for rule systems usage by businesses, business policies and business rules are defined below.

Business Policy Business policies are statements that are used to make decisions. They are typically found inside application code in the form of if-then statements.

Business Rule A business rule is a declarative statement that applies to a business. It is based on a business policies vocabulary and written in understandable natural language.

Here is an example of business policy: Customers who spend a lot of money in a single transaction need an upgrade. One implementation of this business policy could be the following business rule:

```
if      the customer's category is Gold
   and  the value of the customer's shopping cart is more than 1500
then    change the customer's category to Platinum
```

A complete business logic can be defined with a set of rules (called ruleset) to implement any business policy. Business rules are managed in Business Rule Management Systems (BRMS).

1.1.3 BRMS

The information provided in this section is based on IBM Operational Decision Manager documentation [IBM, 2021].

Business Rules Management Systems (BRMS), such as IBM Operational Decision Manager (ODM), provide a business-user-friendly solution for authoring, orchestrating, testing, deploying, and executing business rules. Using a human readable language facilitates translation of decision-making and business strategies into an executable project without requiring any programming skills. Additionally, various tools are provided to help guarantee a proper definition, validity, and solidity of a system. We can mention for example simulations features to test multiples scenarios.

BRMS are based on the principle that business policies should be extracted from application code. By removing the business policy implementation from the application code and implementing business policies with externalized business rules in a BRMS, application code and business logic become independent and a business policy change will not require changes in application code.

1.2 Motivations

Data is being generated at an unprecedented rate, and without the ability to extract meaningful and interpretable information from it, data is significantly less valuable.

Extracting expressive rules from data could provide new and relevant business rules to the BRMS user but also more generally an interpretable method for high-stakes decision-making where trust is provided by transparency and interpretability.

State-of-the-art rule-based models currently have great difficulties learning complex data patterns. Rule-based approaches typically tend to overfit complex patterns because of the inappropriate simplicity of the existing rule languages (limited expressivity [Fürnkranz and Kliegr, 2015]). The grammars of the learned conditions are usually comparable to the grammar of DNF, that is disjunction of conjunctions over binary attributes. This is very limited compared to basic BRMS grammars which include support for numerical data, aggregations, sequential dependencies... As a consequence, the high dimensionality of overfitted models makes human understanding of the model much harder. We believe that having models with more complex (still interpretable) possible data representations will enhance both the interpretability and accuracy of rule models.

Different approaches have been used to extend the rule grammar of learned rules or to find complex symbolic expressions to model data. Neuro-symbolic approaches by combining logical formulae and neural networks can model complex functions [Udrescu and Tegmark, 2020] or provide 1-to-1 mapping with logical formulae [Riegel et al., 2020]. For sequential covering rule learning algorithms (logical methods), like RIPPER [Cohen, 1995] for instance, numerical attributes are required to be converted into nominal attributes. Discretization is usually applied as a preprocessing step and impacts the final rules expressivity [Stańczyk et al., 2020]. For decision tree learning algorithms like C4.5 and C5.0 [Quinlan, 1993, Quinlan, 1996], continuous and discrete attributes are handled and extensions have been proposed to learn temporal and causal rules [Karimi and Hamilton, 2010].

Support for time-dependent and sequential data is crucial for many applications in machine learning. Time series data, such as financial transactions, weather patterns, or sensor readings, as well as sequential data such as DNA sequences, are prevalent in real-world scenarios. The ability to analyze and predict trends in such data in an interpretable manner is of great interest.

To address the issue of rule expressivity and grammar limitations, we will first consider the use of preprocessing and then primarily focus on the usage of interpretable rule-based neural networks.

1.3 Running Example: Fraud Detection

To illustrate the propositions made in this thesis, we will provide examples based on a fraud detection use case. Fraud detection is widely dealt with rule systems for transparency and interpretability reasons (explanations required for the final user) but also because experts can design rules based on their knowledge of the current regulations, and common fraud schemes. Fraud detection is an intuitive and relatable problem that is easy to grasp. It involves complex patterns that are great to demonstrate the capabilities of the proposed models.

However our goal in this thesis is not to learn rules exclusively for fraud detection thus the characteristics specific to this example such as highly unbalanced datasets are not studied here. Instead, it is used as a motivating example that has some shared characteristics with other applications. This thesis aims to provide insights and potential solutions for any decision making problems where learning rules is of interest. We can mention for instance loan approval, churn prediction or disease diagnosis.

It is important to note that the examples of fraud patterns used in the manuscript are purely hypothetical. They do not represent real-world fraud

Account		
id	category	the id of the account
isNewAccount	binary	the customer's account was recently created
accountBalance	number	the balance of the account
accountCountry	category	the country of the account
Transaction		
isDestAccountSusp	binary	the destination account is labeled as suspicious
isHighRiskCountry	binary	the transaction occurs in a high-risk country
amount	number	the amount of the transaction
oriAccount	Account	the origin account of the transaction
destAccount	Account	the destination account of the transaction

Table 1.1: Running example object model. Fraud detection patterns will be based on transaction and account attributes that are either binary, numerical or categorical. We will assume the *account id* possible values to be a set of unique ids.

patterns or real data. The object model on which the examples will be based on is presented in Table 1.1.

1.4 Outline

This thesis presents a novel approach for learning expressive rules using neural networks. It is organized into two main parts. Part one consists of a review of the state of the art and some preliminary work around the use of a preprocessing approach to learn more expressive rules. Part two of the thesis is comprised of three chapters presenting models of increasing complexity evaluated on new synthetic benchmarks. The first one, chapter three, introduces the base model: a neural-based rule learning model and its use with binary, numerical and categorical data. Chapter four extends this model to sequential data, introducing a neural-based rule learning model for temporal data. Finally, chapter five presents an extension with a deeper architecture that incorporates patterns with predefined aggregation functions, allowing for the extraction of more complex knowledge from the data.

Overall, the thesis aims to demonstrate the potential of neural-based rule learning models as a powerful tool for end-to-end expressive classification rule learning.

Part I

State-of-the-Art and Preliminary Work

Chapter 2

State-of-the-Art

Contents

2.1 Rule Systems

The information provided in this section is based on IBM Operational Decision Manager documentation [IBM, 2021].

2.1.1 BRMS

As introduced in the previous chapter, Business Rules Management Systems (BRMS) provide a solution for companies to adapt their business policies and decision-making strategy in a flexible environment. A complete business logic can be defined with a set of rules (called ruleset) to implement any business policy. Thanks to the technical and business separation, business logic can be packaged and integrated into the application code as a decision service. Both live independently: a business policy change will not require changes in application code.

BRMS have numerous operational and strategic benefits inherited from this separation.

- Focus on decision making: BRMS are code-free solutions that prevent business users from relying on IT experts and technical implementation. They can focus on what they are experts in: decision making.
- Monitoring decisions.
- Ability to deal with change: changes are trackable, and lifecycle is made easier with project versioning.
- Risk management: By automating processes in an externalized, unambiguous, monitored, and tested environment the risk is minimized. The ability to monitor the historical modifications of the models as well as the coordination facility between multiple experts highly contribute to that matter.

Specific features make BRMS appropriate for business users, here we list some of the key business-user-friendly features.

User-friendly interface BRMS provide comprehensible graphical interface designed especially for the business user as well as intelligent completion and integrated helping tools.

Multiple rule definition methods There are multiple types of rules to define business rules. Action rules and decision tables are the most common ones.

Rule orchestration To specify how the rules are related to one another, BRMS provide a structure called ruleflow. It provides the ability to orchestrate a complex problem into tasks and subtasks in a tree architecture.

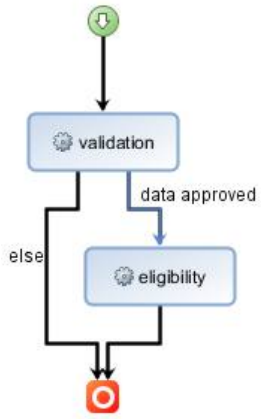


Figure 2.1: Example of ruleflow.

Testing tools Ruleset testing solutions are available to validate the correct behaviour of a rule project.

Lifecycle management and Execution Server Rule project can be monitored and deployed to a server, performances analysed and problems troubleshooted via intuitive interfaces.

While all those features are crucial to a BRMS, the rule language and the rule engine are its main components.

2.1.2 Rule Language: the example of ODM

Each BRMS has its own environment and therefore a different rule language. In this section, we will focus on the example of IBM Operational Decision Manager (ODM). There are three different rule languages in ODM:

- Business Action Language (BAL), which provides a simple if-then syntax to write business rules.
- ILOG Rule Language (IRL), which is similar to Java [Arnold et al., 2005] code, and used for technical rules with complex loops in action part of the rules.
- Advanced Rule Language (ARL), which is a read-only rule language that is similar in syntax to Java 7 and used by the decision engine for execution. So for each BAL expression there is a generated ARL equivalent expression.

The rule language to build and write rules and thus the one to consider for rule learning is BAL. It is a structured human-readable language which is very close to natural language as shown in Equation 2.1.

```
if    the amount of 'the loan' is more than 1,000,000
then  add "The loan cannot exceed 1,000,000" to the messages of 'the loan';
      reject 'the loan' ;
```
(2.1)

BAL provides a simple if-then-else syntax (Equation 2.1) as well as a definition, aggregation, and complex conditions construction syntax. To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else. The example in Equation 2.2 shows the four parts in an action rule with more complex conditions. For more details on the BAL grammar refer to IBM documentation [IBM, 2021].

```
definitions
      set applicant to a customer where the category of this customer is Gold
if          the value of the applicant's shopping cart is more than $100
      or    there are 10 customers where the category of each customer is Gold
then  apply a 15% discount
else  apply a 5% discount
```
(2.2)

The language combined with a defined vocabulary is used to write business rules. This vocabulary is called Business Object Model (BOM). It is an object model that contains classes and methods a rule can act on. In practice, it is very similar to a Java object model. It consists of classes grouped into packages. Each class has a set of attributes, methods and, possibly, other nested classes. BOM is illustrated in Figure 2.2.

BOM has a corresponding executable model called Execution Object Model (XOM) defined by a BOM-to-XOM mapping illustrated in Figure 2.3. Every BOM element must have a corresponding XOM element. This mapping is the key to having a natural-language-based interaction with the user and still have an executable model behind.

Once rules are written, an engine is required to specify how the rules need to be executed. The rule engine is the second key component of a BRMS.

2.1.3 Rule Engine: the example of ODM

Each BRMS has its own rule execution solutions. The execution mode affects which rules are executed and in which order. ODM provides multiple engine execution modes: RetePlus, Sequential and Fastpath modes. They all have benefits and drawbacks which makes them more or less adapted for a particular rule task. Here we list how they operate.

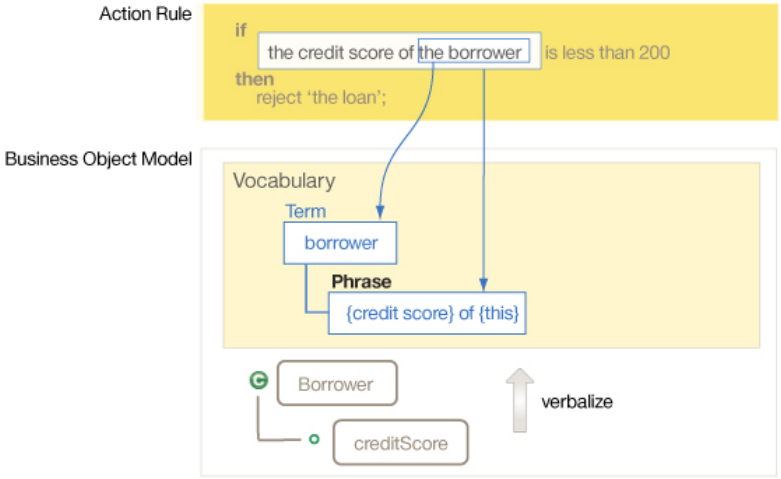


Figure 2.2: Business Object Model and Action Rule.

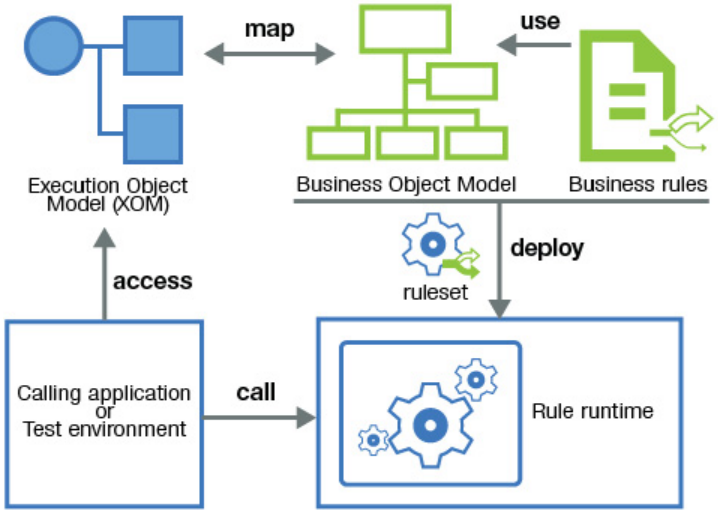


Figure 2.3: BOM and XOM.

RetePlus mode [IBM, 2021] RetePlus mode is an extension of the Rete algorithm [Forgy, 1982] and operates as follows.

1. The rule engine matches the conditions of the rules in the ruleset against the objects in working memory as shown in Figure 2.4. During the pattern matching process, RetePlus creates a network based on semantic relationships between rule condition tests.

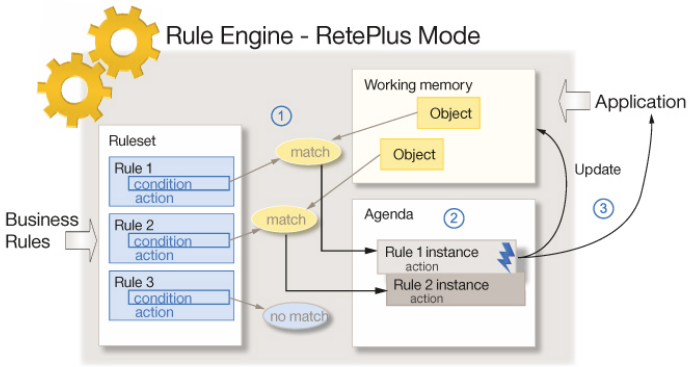


Figure 2.4: RetePlus mode.

2. For each match, a rule instance is created and put into the agenda. Then, based on some ordering principles, the agenda selects the rule instance to be run.
3. When the rule instance is executed, the rule action is executed. This action modifies the working memory in the following way
   - By adding an object to the working memory.
   - By removing an object from the working memory.
   - By modifying the attributes of an existing object.
4. The process is repeated until no more rule instances are left in the agenda.

Sequential mode [IBM, 2021] In sequential mode, rules are stateless, and executed in ruleset order like a stack as shown in Figure 2.5. There is no reevaluation of a rule, so conditions with aggregations are not supported (there is at least one element in a condition for example). However, workarounds exist using the working memory.

The sequential algorithm operates as follows:

1. The rule engine does pattern matching on input ruleset parameters and on the conditions defined on the collections of objects in working memory.
2. For each match, a rule instance is created and immediately run. When a rule instance is run, it sets the value of an attribute or an output ruleset parameter.

Fastpath mode (default mode) [IBM, 2021] The Fastpath algorithm is illustrated in Figure 2.6 and operates as follows:

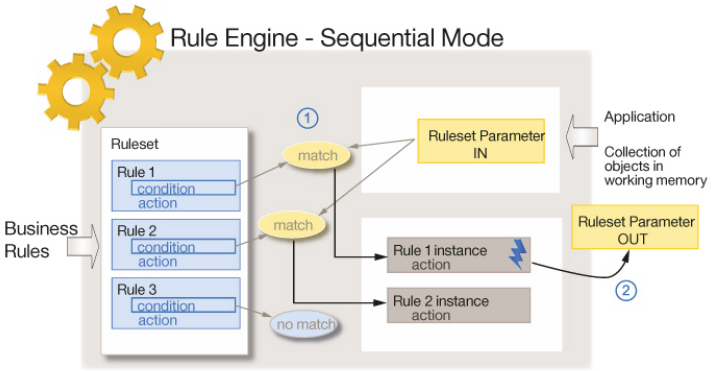


Figure 2.5: Sequential mode.

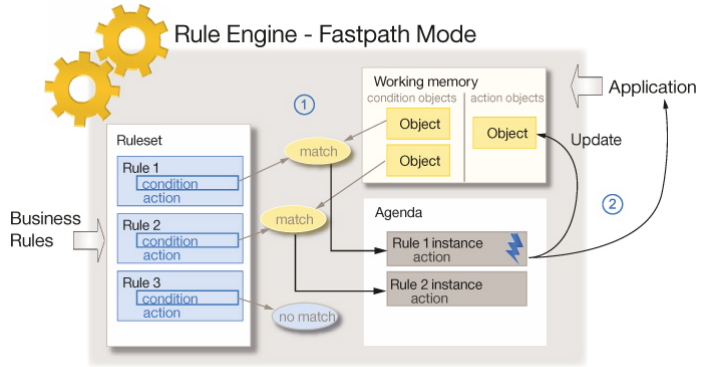


Figure 2.6: Fastpath mode.

1. The rule engine uses a working memory that references application objects or ruleset parameters. Fastpath does the pattern matching process, as in RetePlus, by creating a tree based on semantic relations between rule condition tests.
2. For each match, a rule instance is created and inserted in the agenda.
3. After the pattern matching process, the rule instances in the agenda are executed.
4. The rule engine stops after the rule instances have been executed. This behaviour also depends on the exit criteria of the rule task. The pattern matching process is not repeated.

To conclude, in this section, we described the basics of how a BRMS works thanks to its environment and main components that are its rule language

and rule engine. While BRMS suit the businesses needs in terms of flexibility, efficiency and various data types support, in a world where machine learning is solving the most complex problems and data is everywhere, a feature is highly requested: learning business rules from data. Rules are commonly written by business users. However, learning rules from data brings a new dimension to rule systems: expertise combined with learning abilities.

2.2 Machine Learning Basis

In this section, we define some key machine learning concepts that are required for further reading.

Machine learning algorithms aim to create a predictive or decision-making model by analyzing a dataset. The model is designed to make predictions or decisions based on new input data. A dataset is characterized by attributes and a set of instances. An instance assigns a value to each one of the attributes. An instance is commonly also referred to as an observation, an example, or an item. Depending on the properties of the dataset, it can be used for different learning tasks and scenarios. For a supervised learning scenario, - i.e. when learning a model that maps an input to an output based on example of input-output pairs - one of the attributes is a target or output attribute. Attributes that are not target attribute are commonly called features. For an unsupervised learning scenario, there is no target attribute in the dataset. For a classification learning task, the goal is to assign a category to an instance whereas for a regression learning task the outcome is continuous.

In the following, we will focus on supervised classification learning problems. In that context, the target attribute is often called class attribute or label.

2.3 Rule Learning

In machine learning, rule learning is a procedure of generating rules from either existing rules or models and/or data. Rule learning includes various types of inferences, such as inductive, deductive, and analogical reasoning, though rule induction is the most common one [Wojtusiak, 2012].

Rule induction, or inductive rule learning, is the process of automatically generating a set of rules (a ruleset) from a dataset. The learned ruleset is a model that is intended to be both readable and comprehensible by a human, and that can be executed to make a prediction or a decision.

Thanks to the specific features of a ruleset model, rule learning is considered for two applications: learning and explaining. They come hand in hand as a learned ruleset is explainable, but rule learning can also be used to explain a decision made by a trained model. This idea was studied by [Sushil et al., 2018] with proposition of different approaches to induce if-then-else rules to explain predictions of supervised models.

In the following, main state-of-the-art rule induction algorithms are presented. As explained by [Fürnkranz et al., 2012], there are two main families of methods to induce rulesets from training data: extracting rules from a decision tree (CART [Breiman et al., 1984], C4.5 [Quinlan, 1993]) or sequential covering that is learning rules directly from data (CN2 [Clark and Niblett, 1989], RIPPERk [Cohen, 1995]). Both are detailed in the following subsections. We also add another family of method of interest: neural-based rule learning, which we will study later in Section 2.4.

In this document, we will focus on rule induction. Thus we will use *rule learning* and *rule induction* interchangeably.

2.3.1 Tree-Based Algorithms

Decision trees are often used to find an optimal solution of a problem. A decision tree is a recursive structure made of labeled leaf nodes or test nodes that have two or more outcomes. In the following example (Figure 2.7), leaf nodes are represented by an ellipse and test nodes by a rectangle.

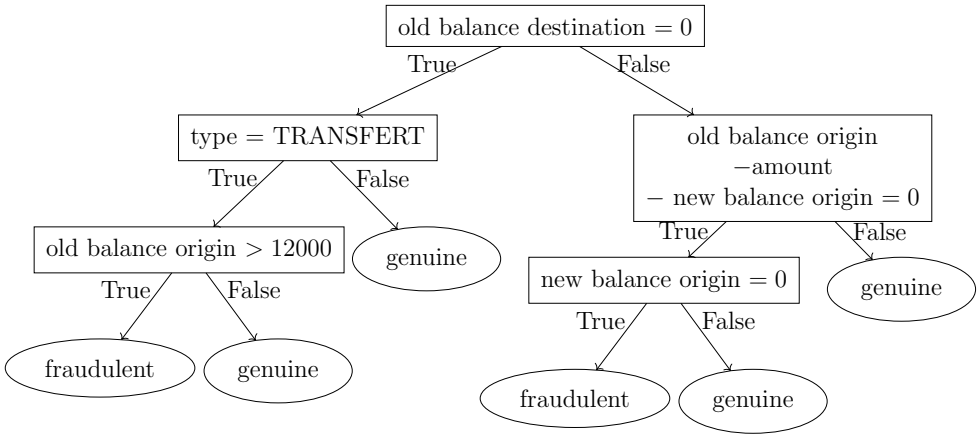


Figure 2.7: Example of a decision tree for financial fraud detection ("Synthetic Financial Datasets For Fraud Detection").

Decision tree learners follow a divide and conquer paradigm where the idea is to decompose the problem into similar and simpler subproblems and

combine the intermediate solutions to build the final one.

When building a decision tree, the main concern is how to choose the value of a dividing node, that could also be referred to as a test node, a condition, or a discriminator.

Any decision tree can be converted into a set of classification rules. A path in a tree, called decision path, can be written as an if-then rule where the outcome of the rule is the leaf node, and the conditions are a conjunction of decision nodes encountered. In the example Figure 2.7, one path can be written as the following rule (Equation 2.3).

```
if          old balance destination = 0
      and   not (type = TRANSFERT)
then  transaction is genuine
```
(2.3)

To describe the entire decision tree in the same manner, as there are 6 different paths, we would need to write 6 rules. However, if we look more closely into the decision tree, different approaches can be taken to simplify the ruleset. Rules could be ordered and a default class defined, for example. We could also introduce an *else* clause and the use of disjunctions. One solution is presented in Equation 2.4. A complete chapter in Quinlan's C4.5 book [Quinlan, 1993] is dedicated to the study of converting a decision tree into rules.

```
if                old balance destination = 0
          and     type = TRANSFERT
          and     old balance origin > 12000
      or          not (old balance destination = 0)
          and     old balance origin - amount - new balance origin = 0
          and      new balance origin = 0
then  transaction is fraudulent
else  transaction is genuine
```
(2.4)

Multiple methods have been proposed to induce decision trees from data such as CART (Classification And Regression Trees) [Breiman et al., 1984] or C4.5 [Quinlan, 1993]. C4.5 is an algorithm proposed by Quinlan to generate decision trees [Quinlan, 1993]. In this algorithm the quality evaluation of a condition or test is based on information theory with information gain and a gain ratio criterion.

2.3.2 Sequential Learning

Sequential learning (not to be mistaken with learning from sequential data), sequential covering and separate-and-conquer strategy all refer to the same

idea: learning a disjunctive set of rules one rule at a time. By analogy with decision trees "divide and conquer" paradigm, some rule induction algorithms are based on a similar idea called "separate and conquer". A separate and conquer algorithm searches for a rule that describes a part of the training instances, separates the data and, with a recursive approach, it conquers all the remaining examples not covered by any new rules until there is no more training data. Multiple induction algorithms are based on this strategy.

CN2 for example is a reference model based on this strategy [Clark and Niblett, 1989]. Algorithms are also intended for specific input data. For example, FOIL (First-Order Inductive Learner) [Quinlan, 1993] is a sequential learning model that learns Horn clauses from relational data.

One of the current state-of-the-art reference algorithm is RIPPERk (also refereed to as RIPPER) [Cohen, 1995]. It also follows the "separate and conquer" strategy. RIPPER is an algorithm presented in 1995 by W. Cohen. It is an improved version of the IREP (Incremental Reduced Error Pruning) algorithm presented in Algorithm 1 for two classes learning problems [Fürnkranz and Widmer, 1996].

Algorithm 1: IREP algorithm for a two-class learning problem

```
Input: Pos, Positive elements in a dataset
Input: Neg, Negative elements in a dataset
Output: RuleSet, Set of rules learned
RuleSet empty;
while Pos not empty do
    Split (Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos,
     PruneNeg);
    Rule = GrowRule(GrowPos, GrowNeg);
    Rule = PruneRule(Rule, PrunePos, PruneNeg);
    if error rate of Rule on (PrunePos, PruneNeg) > 50% then
        return RuleSet;
    else
        Add Rule to RuleSet;
        Remove examples covered by Rule from (Pos,Neg)
    end
end
Return RuleSet
```

In Algorithm 1, *GrowRule* refers to the "growing of a rule" which is implemented by [Cohen, 1995] such that it will "repeatedly add conditions that maximizes FOIL information gain criterion until the rule covers no negative

examples from the growing dataset". FOIL information gain is defined in Equation 2.5.

$$G_{\text{foil}}(rule_0, rule_1) = p_1(\log_2(\frac{p_1}{p_1 + n_1}) - \log_2(\frac{p_0}{p_0 + n_0})) \tag{2.5}$$

where: $rule_0$ = rule before adding new condition
$rule_1$ = $rule_0$ with new condition
p_0, n_0 = number of positive (and negative) elements covered by $rule_0$
p_1, n_1 = number of positive (and negative) elements covered by $rule_1$

In Algorithm 1, *PruneRule* consists in "deleting any final sequence of conditions from the rule, and, choosing, the deletion that maximizes" the following function (Equation 2.6).

$$cost(Rule, PrunePos, PruneNeg) = \frac{p + (N - n)}{P + N} \tag{2.6}$$

where: p, n = number of positive (and negative) elements in $PrunePos$ ($PruneNeg$) covered by $Rule$
P, N = number of positive (and negative) elements in $PrunePos$ ($PruneNeg$)

As explained in [Cohen, 1995], this process is repeated until the value of the cost function can not be improved by deletion of conditions.

Three modifications are made to IREP algorithm to create RIPPER.

1. **Metric used in pruning phase** changed from cost function described in Equation 2.6 to cost function defined in Equation 2.7 because first cost function tends to prefer more coverage rather than accuracy.

$$cost(Rule, PrunePos, PruneNeg) = \frac{p - n}{p + n} \tag{2.7}$$

2. **Stopping condition** is too restrictive in IREP and often stops too soon for moderate size datasets and low coverage rules. Minimum Description Length (MDL) is introduced, and the following condition (Equation 2.8) replaces the former error-rate-based condition.

$$\text{DL of } RuleSet > d + \text{MDL obtained so far} \tag{2.8}$$

MDL refers to the number of bits needed to encode theory data from which it was learned. Common value for d is 64 bits.

3. **Post processing to optimize the rules** is added to minimize error on complete ruleset. For each rule in the ruleset, two new rules are learned. The first one is called replacement rule, and it is learned from scratch with the data covered by the original rule. The second one is call revision rule, it is learned starting from the original rule and by adding new conditions with the data covered by the original rule. Both rule creations follow the same growing and pruning approach but this time pruning aims to minimize error on the entire associated ruleset. Only the rule with the lowest MDL between original, replacement and revision rules, is kept as shown in Algorithm 2. This post-processing of the ruleset can be done k times, which is referred to by the algorithm full name RIPPERk.

Algorithm 2: RIPPERk Ruleset post-processing

```
for k times do
    foreach rule R_i in ruleset do
        create Replacement rule R_i' → ruleset'
        create Revision rule R_i'' → ruleset''
        choose R_i, R_i' or R_i'' :
        R_i = rule associated to ruleset with lowest MDL
        Update ruleset
    end
end
```

We detailed IREP and RIPPER algorithms to point out how complicated it is to grasp how they deal with the quality and rule complexity tradeoff. Although they try to optimize this tradeoff, pruning phase, stopping condition combined with post processing for RIPPER make it difficult for the user to configure the learning to obtain more rules for example.

Interesting approaches have been proposed to improve RIPPER performances. A fuzzy rule-based classifier called FURIA was introduced in 2009 in which decision boundaries are modeled in a more flexible way [Hühn and Hüllermeier, 2009]. In that example, improvements come with an increase in runtime.

2.3.3 Associative Rule Classifiers

Another approach to classification rule learning is to combine Association Rule Mining (ARM) with classification.

In association rule learning, rules are defined as an implication of the form $X \rightarrow Y$ where X and Y are sets of binary attributes, also called items. To select the best rules in a learning problem, constraints need to be defined to shrink the space of all possible rules. Those constraints are commonly minimum support and minimum confidence. Support of X is the proportion of instances in the dataset containing all items of X. It indicates how frequent X appears in the data. Confidence of a rule $X \rightarrow Y$ is the proportion of instances in the dataset containing all items of X and all items of Y. It indicates how frequent the rule $X \rightarrow Y$ is true in the dataset.

The first associative rule classifier (ARC) algorithm was introduced by [Liu et al., 1998] with CBA algorithm (Classification Based on Associations) which generates a complete set of CARs (Class Association Rules) [Liu et al., 1998]. It combines a state-of the-art association rule learning algorithm Apriori [Agrawal and Srikant, 1994] with some interesting ideas taken from both research fields (i.e. ARM and classification rule learning) to provide better results than C4.5. One presented explanation to this result is that "global" best rules are created rather than "local" ones in a sequential covering approach. "Local" best rules are according to Liu et al. more likely to be overfitting. This improvement can be explained by the ability of the model to express patterns or expressions (expressiveness) that are relevant for the data considered.

Multiple propositions have been made to improve CBA algorithm. ACIM (Associative Classification based on Incremental Mining) for example, reduces computational time while maintaining comparable accuracies [Alnababteh et al., 2014].

CPAR (Classification based on Predictive Association Rules) comes with interesting characteristics [Yin and Han, 2003]. In comparison to RIPPER that evaluates all candidates rules, it uses a more efficient greedy approach for generation. Additionally, repeated calculations are limited with dynamic programming and multiples rules are built simultaneously.

All previously mentioned associative classification algorithms share the common requirement presented above: thresholds (commonly based on confidence and support) need to be provided by the user for rule mining. Setting these values is a challenging task as it differs for different applications.

Recently, a bi-level associative classifier using automatic learning on rules called BiLevCSS tried to tackle this problem and outperforms state-of-the-art classifiers (in terms of accuracy) [Sood et al., 2020]. It is based on two learning stages. First an associative rule classifier is built based on statistically significant rules which is highly inspired by SigDirect algorithm [Li and Zaiane, 2017]. This classifier is then followed by a supervised learning classifier in the second stage of learning. Multiple classifiers are considered for

this second step, but we can highlight the use of RIPPER for interpretability reasons.

2.3.4 Sequence Classification

Rule learning can also be applied to the problem of classifying sequences. Sequence classification has been an important research problem in machine learning because of the wide range of applications from genomics to any time series data. While there are now impressive black-box models capable of solving this problem with high accuracy, interpretable methods are still of great interest.

A sequence is an ordered list of events from which several subtypes can be defined [Xing et al., 2010].

- a *simple symbolic sequence* is an ordered list of the symbols from an alphabet of symbols (a DNA sequence for example),
- a *complex symbolic sequence* is an ordered list of vectors. Each vector is a subset of an alphabet. For example, a set of items bought by a customer over a certain amount of time,
- a *simple time series* is a sequence of real values ordered in timestamp ascending order,
- a *multivariate time series* is a sequence of numerical vectors.

In the context of rule learning on sequential data all of the above can be considered but in practice we have mostly found work on simple symbolic sequences in the literature .

There are methods for instance that first extract interesting patterns before building a classifier [Zhou et al., 2013, Zhou et al., 2015]. Many different types of patterns can be defined but as pointed out by [Zhou et al., 2015], common sequence classifiers are either great for datasets where 'the class of a sequence is determined by certain items that co-occur within it, though not always in the same order' or where 'the class of a sequence is determined by items that occur in the sequence almost always in exactly the same order'. This observation highlights the limitation in terms of sequential expressivity of the existing methods.

MiSeRe was introduced to tackle two other limitations of pattern-based methods for classification problems on sequential data, which are parameter tuning and robustness [Egho et al., 2015]. However it is also a two-step method where any classifier could be used after the patterns have been extracted.

Learning from sequential data comes with multiples challenges. First, capturing the temporal dependencies in the data is not trivial, especially for long sequences, when dependencies are complex or with long-term correla-

tions. On that note, an interesting non interpretable approach for capturing long-term correlations was recently introduced based on a rotation layer and an innovative data representation [Khalitov et al., 2023]. Then, the dimensionality of the data even after having extracted potential sequential features can still be very high. Finally, common machine learning methods work with fixed length input size i.e. fixed sequence length, when in practice sequential data often comes in variable-length sequences.

To conclude, in this section, we introduced state-of-the-art rule learning methods that have been extensively used and studied in the past. While they address the problem of learning logical operations like disjunctions, conjunctions or implications or the learning on symbolic sequences, the grammar of the learned rules is very limited compared to the grammar of any BRMS.

2.4 Rule Learning with Neural Networks

In this section, we will start by studying different approaches combining logical and neural aspects of learning in neuro-symbolic methods and identifying those that are relevant for rule learning. Then, we will focus on two categories. The first one, rule extraction, consists in extracting rules from trained neural networks. The second one, a specific sub-section of the literature that is more relevant to this thesis, is related to enforcing logical structures in a neural network architecture to make the models equivalent to logical models.

2.4.1 Neuro-symbolic

Neuro-symbolic Artificial Intelligence (AI) combines neural and symbolic architectures with the aim of benefiting from both domain. Recently, a lot of research interest has been focused on this topic, partly because more and more researchers believe that a human-level AI requires symbolic reasoning but also for robustness, reliability or interpretability [Marcus and Davis, 2019, Sarker et al., 2021, d'Avila Garcez et al., 2019].

Different surveys have recently been published to clarify the challenges, existing methods, applications and future directions of the broad field of neuro-symbolic AI [d'Avila Garcez and Lamb, 2023, Yu et al., 2023]. Different taxonomies are presented and their complexity highlight the variety of the different existing approaches. For instance, [Kautz, 2022] proposed to divide neuro-symbolic approaches into 6 categories.

The first one refers to 'standard deep learning' as symbols can serve as both the input and the output of neural networks. For instance, in the context of natural language processing, the symbolic inputs are sequences of

words that can be converted into vectors with one-hot encoding for example. For the output of the neural network, depending of the task, there are scenarios where the output can be symbolic. For example, again in the context of natural language processing, outputs can be sequences of symbols for text generation or symbolic category for sentiment analysis. This category is very broad and somewhat apart, because in most cases, when the field of neuro-symbolic AI is mentioned, it is to differentiate it from 'standard deep learning' and thus it is not this type that is being referred to.

The second category gathers systems that use neural-based techniques within a symbolic problem solver (like Monte-Carlo Tree Search algorithm). This is the case for AlphaGo for example where 'value networks' (evaluate board positions) and 'policy networks' (select moves) are combined in a Monte-Carlo Tree Search algorithm that selects actions by lookahead search [Silver et al., 2016].

The third type is for neural networks that transform non-symbolic input into a symbolic data structure before it is given to symbolic reasoning model i.e. input/output interactions between the two modules. DeepProbLog for instance, in the context of probabilistic logic programming, is based on this idea with neural networks being predicates that are then called by the probabilistic logic program [Manhaeve et al., 2018].

In the fourth category, systems have the particularity to have the symbolic knowledge compiled in the training of the neural network i.e. symbolic aspects are translated into the architecture of the network. In this category, we find differentiable fuzzy logic where weights are constrained to act as logic operators like AND and OR [Riegel et al., 2020, Qiao et al., 2021] but also neural approaches addressing symbolic problems like performing calculations [Lample and Charton, 2019]. In the latter, a transformer is trained for function integration and for solving differential equations on large datasets of equations, with their associated solutions.

The fifth type groups systems that use soft-constraints on the network's loss function to apply a symbolic logic rule constraint on the learned embedding [Diligenti et al., 2017, van Krieken et al., 2022, Petersen et al., 2022]. Prior background knowledge when described in logic rules can for instance be used with this technique to help training with unlabeled and/or noisy data. In that context different fuzzy logic operators have been compared [van Krieken et al., 2022] and it highlighted that some are not suitable in differentiable learning setting.

And finally for the sixth category, the idea is to 'embed a symbolic reasoning engine inside a neural engine', as some of the fourth type systems do, but with the additional goal of 'enabling super-neuro and combinatorial reasoning' according to [Kautz, 2022]. It is based on the distinction between two

types of human thinking defined by [Kahneman, 2011]: slow and fast thinking. The latter is very quick, effortless, involuntary, automatic and based on similarity, like for recognizing faces, and humans are very good at it. The other is slow, demanding, requires awareness and often working memory and is used for example for complex mental calculation but the humans are more likely to make mistakes. A parallel between neural based systems and fast thinking as well as between logical based systems and slow thinking is proposed by [Kautz, 2022] to define the sixth type. As for human reasoning, fast thinking is in charge of initiating slow thinking when needed, the symbolic reasoning part would be a subroutine of the overall neural system. Back and forth interactions between the two would require, the neural network to support both the symbolic reasoning system output and the potential non symbolic initial input data. However as stated by [d'Avila Garcez and Lamb, 2023] although there is work in this direction [Cameron et al., 2020, Lamb et al., 2021], type 6 that are fully embedded systems and capable of complex calculation do not exist yet.

Neuro-symbolic reasoning has also been categorized into 3 subfields based on the manner in which the learning (neural) and reasoning (symbolic) components are coupled : 'learning for reasoning' and 'reasoning for learning' where modules are loosely coupled and 'learning-reasoning' where they are tightly coupled [Yu et al., 2023]. For 'learning for reasoning' systems, the objective is to harness the power of neural networks to improve the performance of symbolic reasoning. For 'reasoning for learning' systems, it is the other way around, the objective being to benefit from symbolic knowledge to constrain or guide the learning for performance or interpretability reasons. Finally for 'learning-reasoning' systems, the neural and symbolic reasoning systems both benefit equally from each other, and the interaction is bidirectional. These categories can be mapped to the previously described types from [Kautz, 2022] as shown in Table 2.4.1 with common examples.

Although all strategies can be of interest in the context of rule learning, most of them are commonly not aiming for fully interpretable models or for symbolic reasoning. In Kautz taxonomy, type 1 for instance is too broad to be considered as an interesting category for rule learning. Types 2 and 3 are not commonly referring to fully interpretable strategies because they rely on intermediate concepts that are not. For strategies in type 5, soft-constraints are used and thus symbolic constraints are met only in exceptional cases. However for strategies in types 4 and 6, in cases where there is a 1-to-1 mapping between the network and logic, an application for rule learning appears to be natural. In the following, we dive more in depth into how neuro-symbolic approaches can be used for rule learning.

Taxonomy Yu et al.	Taxonomy Kautz	Description	Examples
N/A	type 1	'standard deep learning' with symbolic input/output of a neural network. ⊖ *commonly not interpretable, not learning rules*	
Learning for reasoning	type 4	symbolic embedded in the neural architecture of the network. ⊕ *when 1-to-1 mapping, ensures logic expression learned* ⊖ *limited expressivity*	[Rocktäschel and Riedel, 2017] [Evans and Grefenstette, 2018] [Riegel et al., 2020] [Qiao et al., 2021] [Persia and Guimarães, 2023]
	type 2	neural-based techniques used within a symbolic problem solver. ⊖ *commonly not fully interpretable, not learning rules*	[Silver et al., 2016] and autonomous vehicles systems
Reasoning for learning	type 5	symbolic constraining the training of the model (e.g. in loss). ⊖ *soft-constraints, logic is an exception*	[Diligenti et al., 2017] [van Krieken et al., 2022] [Petersen et al., 2022]
Learning-reasoning	type 3	input/output interactions between a neural network and a symbolic reasoning model. ⊖ *commonly not fully interpretable, not learning rules*	[Manhaeve et al., 2018] [Barbiero et al., 2023]
	type 6	symbolic reasoning engine embedded inside a neural engine aiming for combinatorial reasoning. ⊖ *no existing method yet*	in the direction: [Cameron et al., 2020] [Lamb et al., 2021]

Table 2.1: Different taxonomies [Yu et al., 2023, Kautz, 2022] for the field of Neuro-Symbolic AI illustrated with examples and contextualized in the context of rule learning in italics.

2.4.2 Rule Extraction from Intermediate Models

An approach to rule learning is to extract rules from an intermediate model (e.g. a deep neural network). The resulting rules can either be used as an explanation of the model, as an approximation of the model or as the final model. This strategy corresponds in the previously described taxonomies to a 'learning for reasoning' and type 2 strategies. It corresponds to a sequential technique where first a neural network is trained and then a rule extraction technique is applied. The objective is therefore, given a trained model and the data used for training, to extract the best comprehensible rules to approximate the model.

As reviewed by [Hailesilassie, 2016], there are three main extraction methods. Decomposition algorithms work at the neuron level and aggregate the results to build a network representation. Pedagogical approaches consider the neural network as a black-box, so the internal weights are not studied. Eclectic methods combine the two previous methods. We can mention DeepRED [Zilke et al., 2016] that first presented a decomposition method for deep neural networks and CGX a method using dual linear programming [Hemker et al., 2023]. An interesting pipeline was also presented recently that starts from a trained neural network that is then converted into a random forest and then to a boolean directed acyclic graph [Brudermueller et al., 2020]. This work highlighted that the process of converting a neural network into logic resulted in a decrease in accuracy.

Indeed, with extraction based approaches, the approximation errors of the black-box model are propagated from the trained model to the rules and are combined with the extraction errors. Also, all extraction methods are limited in terms of rule grammar by either the architecture of the model for decomposition algorithms, by the rule learning algorithm used for pedagogical approaches or both for eclectic methods. Rule extraction methods are therefore great solutions for cross checking neural networks but limited for rule learning as the final goal.

2.4.3 Logical Neural Structures

With the success of deep neural networks, few approaches have been proposed to explicitly enforce a logical structure in a neural network.

First we can mention models based on layers that model logical operators like AND and OR. Decision Rule Network [Qiao et al., 2021] for instance is based on differentiable AND/OR layers to learn a disjunction of conjunctions (DNF) from binary input data with parameterized trade-off between accuracy and simplicity of the rules. Binary weights are trained to select the

input nodes to keep in the associated operation (conjunction or disjunction). Also in a general context, deepening the architecture of such neural-based rule learning model was investigated recently and showed promising results in terms of accuracy [Beck and Fürnkranz, 2021b]. However, it is important to note that with such deepening, i.e. stacking and/or layers, the expressivity of the rules is not improved as all resulting logical formulas can be converted into an equivalent DNF. These models are based on differentiable logic from the field of fuzzy logic with different T-norms and T-conorms compared in depth in recent work [van Krieken et al., 2022].

A differentiable logic gate network has also been introduced based on differentiable logic. In that case, weights are randomly fixed and what is learned is the logic gate operator to use for each neuron from a set of predefined operations [Petersen et al., 2022]. Also, the network is intrinsically sparse as each neuron has only 2 inputs. Although a logic gate network is very interesting in terms of computational cost and speed at inference time, the learned logical expression is not meant to be concise nor expressive.

With those approaches, there is generally a 1-to-1 mapping between the (trained) model and the rules or the logical expression that can be expressed by extracting the learned parameters of the neural network. This is because by design, we understand how it works before knowing that it works. This is a completely opposite strategy to the growing field of mechanistic interpretability where neural networks are reverse-engineered.

Also, in the context of rule learning, using a neural-based approach can be potentially more robust to data imperfections than sophisticated symbolic algorithms [Hitzler et al., 2022]. Recently, RIDDLE was introduced to target this specific aspect to learn rules on incomplete or uncertain data [Persia and Guimarães, 2023].

We also want to point out the proximity of the logical neural structures based on binary weights with the research field of Binary Neural Networks (BNN). BNNs are deep neural networks where a vast majority of both weights and activations are binary (usually -1, 1) [Geiger and Team, 2020]. This is a specificity that can be found in the models mentioned above. In the context of a BNN, the binarization makes the computation very efficient in terms of memory and evaluation whereas for rules, discretization is the main concern. We can also highlight the fact that these logical neural structures, in addition to being binary, require sparse weights, which similarly allows us to draw a parallel with sparse neural networks and their challenging training [Srinivas et al., 2017, Louizos et al., 2018].

To conclude, in this section, we introduced state-of-the-art rule learning methods that are more or less based on a neural architecture. We started by

looking into the broad field of neuro-symbolic and analyzed existing strategies in the context of rule learning, to explore combining the hard constraints of rules and symbolic reasoning with the power of neural network. Then, two strategies were studied more in depth : rule extraction from a trained model and logical neural structures. We have shown that both approaches come with limitations including the learned rules grammar being very limited while potentially providing models that are more robust to data imperfections (than non neural-based approaches). Rule extraction methods are great solutions for cross checking neural networks but not as suitable for rule learning as a goal due to the propagation of approximation errors. With logical neural structures on the other hand, a 1-to-1 mapping between the trained neural model and the rules can be obtained at the cost of challenging training due to the requirement for binary and sparse weights.

Chapter 3

Preliminary: a preprocessing approach for more expressive rules

Contents

3.1 Introduction

In order to familiarize ourselves with the subject, the state of the art and the problems introduced previously, we investigate increasing the expressiveness of rules obtained with existing models such as RIPPER using preprocessing. In the following we describe the preliminary work conducted, focusing on the use of mutual information.

Rule-based learning models do not reach the same performance as black-box models. They have weak internal data representations due to the fact that the learning algorithms require binary data as input. The discretization of the data is either automated or done manually, ideally by domain experts. When automated, input data is commonly discretized in an unsupervised manner directly without any other feature processing.

After discretization, input data is of the form $A_i = d$, $A_c \geq \theta$ or $A_c \leq \theta$ where A_i is a discrete attribute and d a valid value for A_i, or A_c is a numerical attribute and θ is a value for A_c. Generating features automatically as a preprocessing step before discretization and rule learning could enhance the expressivity of these conditions with linear combination of features for example.

Different approaches have been proposed to automate attribute construction (or feature generation) for data-mining with filter, wrapper and embedded methods [Sondhi, 2009]. We can mention genetic-programming-based [Otero et al., 2003] and regression-based feature learning with AutoLearn [Kaul et al., 2017]. However as pointed out by [Cherrier, 2021], they mostly do not consider dimensional validity of created features and focus on model performance rather than interpretability. Dimensional consistency is required for instance for full interpretability. It can be controlled for example with the introduction of a grammar [Ratle and Sebag, 2001, Cherrier, 2021]. For filtering methods, a candidate created feature is evaluated with a score function to evaluate the utility of the feature. Different functions have been studied in the literature [Guyon and Elisseeff, 2003] such as correlation and information theory criteria. Information gain is often used as a measure of entropy evolution with and without the feature presence at the dataset level.

In the following, we focus on improving the quality of the learned rules, by extending the rule language with an interpretable preprocessing method based on mutual information for generating new meaningful features automatically for rule learning. We propose an approach for the supervised learning of classification rules with interpretable and dimensionally valid complex conditions. The method relies on the following two hypotheses: augmented data space with selected operations does provide more information for the learning classification task of the chosen datasets, and this added informa-

tion can be measured by mutual information comparison at feature level. We present a filtering method based on mutual information improvement at the feature level. The main motivation for using a preprocessing filtering approach is to build a generic process applicable to any inductive learning algorithm [Otero et al., 2003].

Preliminary results with RIPPER algorithm [Cohen, 1995] are presented, where we focus on numerical attributes and only combine features with dimensionally valid additions and substractions for feature generation. Due to the constraint on the dimensional decomposition of each dataset and the limited number of considered operations, improvements for all datasets is not expected.

3.2 Feature Generation

The proposed feature generation algorithm for numerical features is divided into two steps that are repeated for a maximum of N iterations, as follows.

1. Select the set of possible new features. A new feature F^* is defined by $F^* = f(F_i, F_j)$ with F_i and F_j two existing features and f an operation from the set of available operations. F^* is a possible new feature if $f(F_i, F_j)$ is dimensionally valid and if there is no equivalent expression of F^* already in feature list (exploiting associativity and commutativity properties of operation f if any).

2. For all F^* in possible new features set, add F^* to features list if

$$I(F^*, y) > \max(I(F_i, y), I(F_j, y)),$$

   with I the mutual information and y the target class. Mutual information I is defined as a measure of the similarity between a feature and the target. In practice, we use scikit-learn function estimate '*sklearn.feature_selection.mutual_info_classif*' [Pedregosa et al., 2011].

3. Stop if no new features have been added.

The first step narrows down the search space for possible feature combinations while making sure that the added features are dimensionally valid. The second step makes sure the added feature is meaningful to the classification problem of interest. However, it is important to note that this step does not take into account the complete set of existing features and the correlations between them. Mutual information quantifies the amount

of information shared between the features and the class to predict and can capture non-linear relationship between the variables.

At each iteration, expressions can grow exponentially in complexity. Without loss of generality, selecting new features only based on combinations of an existing feature with one original features at every iteration can help restraining number of possible expressions and ease the equivalence checking in first step.

3.3 Experiments

Datasets The selected datasets from UCI [Dua and Graff, 2017] are presented in Table 3.1.

Datasets are analyzed and to every numerical feature is assigned a unit to be used for the dimensionality filtering. These units either correspond to continuous or discrete data. The details of these assignments is available in Appendix B.

Name	# F (F_{num})	# instances	# classes
abalone	8(7)	4177	29
adult	14(6)	48842	2
breast-wis(consin)	10(10)	699	2
iris	4(4)	150	3
segment	19(19)	2310	7
wine	13(13)	178	3
wine-(quality-)red	11(11)	1599	11
wine-(quality-)white	11(11)	4898	11

Table 3.1: UCI datasets [Dua and Graff, 2017] and associated number of features F (and numerical features F_{num}), instances and possible classes. Additional references : breast-wisconsin [Bennett and Mangasarian, 1992], wine quality [Cortez et al., 2009].

Synthetic Dataset As a sanity check, a very simple balanced generated dataset of 10000 instances is created to justify the necessity to have conditions based on multiple attributes. (x_0 and x_1 being floats between 0 and 100).

$$\text{if } x_0 + x_1 > 100 \text{ then } class = 1 \text{ else } class = 0$$

Without preprocessing, RIPPER manages to approximate synthetic data with 24 rules and 97% accuracy. With preprocessing, the rule is retrieved and perfect classification is achieved.

Experimental Setting Datasets are cleaned when required by removing null values. The experiments are carried out with k-fold cross-validation strategy with $k = 10$. For feature generation (Section 3.2), available operations are sum and absolute difference. Maximum of iterations N is set to 1 (the datasets are simple and after few tests we noticed that features already carried most of the information required for classification so there were no improvement for $N > 1$). Rule learning model applied after feature generation is RIPPER [Cohen, 1995] algorithm (k=2).

Metrics used to compare experiments are common machine learning metrics (accuracy, balanced accuracy and weighted f1 score) as well as ruleset characteristics (number of rules, number of features not used in rules, number of added features and number of added features used in rules). Metrics are averaged over 10 folds.

We run the experiments on datasets presented in Table 3.1 with and without feature generation.

3.4 Results and Discussion

The results are summarized in Table 3.2. The first observation shown is that, in most cases, the preprocessing step has a positive impact on the number of learned rules. Apart from *adult* dataset, a portion of generated features is used in the learned rules for all datasets. They have on average a positive impact on model metrics for datasets *iris*, *segment* and *wine*. Also we can point out that for *wine-quality-white* (and *wine-quality-red*) dataset, the number of rules learned dropped on average from 29.2 to 14.5 (from 22 to 14). It highlights the fact that preprocessing has, as in any machine learning problem, a huge role to play for rule learning. It can affect radically a result in terms of accuracy and ruleset characteristics but also by increasing the expressivity of the learned rules.

We can see that proposed automated feature generation based on mutual information does provide valuable expressions for the next learning step. We list below some of the created features that are used in the learned rulesets.

- abalone: `length + height`
- iris: `sepal length - sepal width`
- segment: `vedge mean + intensity mean`
- wine: `alcohol + ash`
- wine-quality-red: `fixed acidity - volatile acidity`

They do make explicit sense, and it is thanks to dimensional analysis. This dimensionally-aware strategy does both enhance the search efficiency and the interpretability of the resulting expressions in rule conditions as written by

Dataset		Model			Ruleset		
		acc	bal acc	f1	# rules	A	B
abalone	w/o	24.0	12.3	17.6	47.2	8(8)	–
	w/	**24.8**	**12.2**	17.9	47.6	11.2(11.3)	**3.2**(3.3)
adult	w/o	82.2	69.7	80.6	42.2	10.4(14)	–
	w/	82.2	69.7	80.6	42.9	10.4(16)	**0**(2)
breast-wis	w/o	86.1	80.9	84.9	17.3	8(9)	–
	w/	86.0	80.4	84.6	**14.9**	**12.6**(42.3)	6.4(**33.3**)
iris	w/o	92.0	92.0	91.9	5.7	3.8(4)	–
	w/	**95.3**	**95.3**	95.3	**5.6**	5.2(6)	**1.5**(2)
segment	w/o	69.8	69.8	63.2	25.7	14.2(19)	–
	w/	**70.9**	**70.9**	65.2	**26.1**	18.9(28.3)	**5.7**(9.3)
wine	w/o	88.9	89.4	88.6	5.1	5.4(13)	–
	w/	**90.0**	**90.7**	89.4	**4.5**	**5.9**(34.6)	**2.8**(**21.6**)
wine-red	w/o	53.6	24.9	50.1	22.0	11(11)	–
	w/	47.5	20.8	39.9	**14.3**	**21.6**(49.5)	**13.4**(**38.5**)
wine-white	w/o	50.6	21.5	46.1	29.2	11(11)	–
	w/	50.3	20.0	42.1	**14.5**	**24.6**(57.3)	18.1(**46.3**)

Table 3.2: Average metrics over 10-fold cross-validation comparison with (w/) and without (w/o) preprocessing. Weighted averaging for f1 score. A: number of features used in rules (total number of features). B: number of added features used in rules (number of added features).

[Keijzer and Babovic, 1999]. For example, for *iris* dataset, the following rule (among others) was learned:

$$\text{if petal length} \geq 4.8 \text{ and } \text{sepal length} - \text{sepal width} \geq 3.5 \text{ then } class = \text{Iris-virginica}$$

The condition `sepal length - sepal width` ≥ 3.5 describes the elliptic characteristic of *virginica* sepal, whereas the *versicolor* sepal has more of a circle shape, as confirmed in Figure 3.1.

On the other hand, not all datasets benefit from those added features. In four cases, a lot of features are added with the preprocessing step (*breast-wisconsin*, *wine*, *wine-quality-white* and *wine-quality-red*) but are mostly ($\geq$ 60%) not kept in rules learned by RIPPER. This is a consequence of the independence of the feature generation and the rule learning. New features are created based on their mutual information with the target. This is not a metric or criteria used in RIPPER, decisions made at preprocessing time have no relation with decisions made at learning time. It also shows that preprocess-

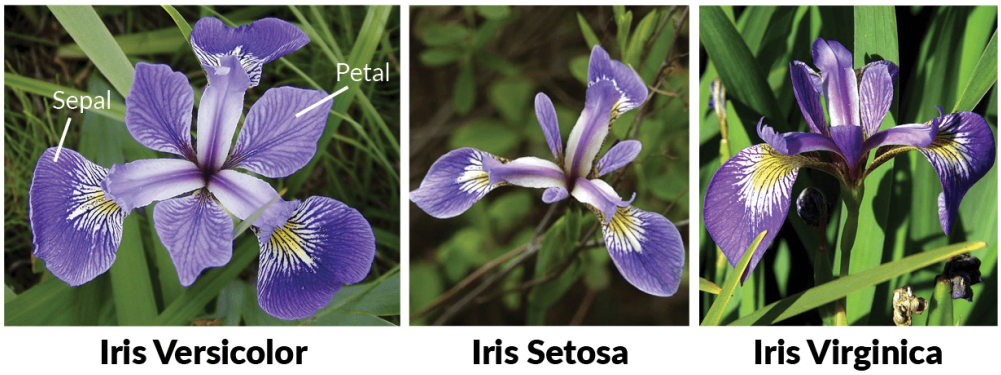


Figure 3.1: Comparison of different types of iris from *iris* dataset (Source: datacamp.com).

ing is not restrictive enough with regards to RIPPER conditions selection for specific datasets. Many features are created and not used in rules. Part of the filtering is carried by the dimensional analysis filter, which is necessary to build comprehensible and valid rules, but it is only a first level filter. This is a common drawback of using filter methods as highlighted by [Cherrier, 2021]: filtering methods are completely independent from the learning step. Using a wrapper method, that is selecting the new features with regards to learning results, would help increase the added feature usage at the cost of computational time. Embedded methods on the other hand, which introduce feature construction in learning algorithm, are not model-agnostic and require different changes for each learning algorithm. This simple method, has the advantage to be applicable for any inductive learning algorithm.

Also, the rule learning model RIPPER is affected by the number of attributes provided as input during training. The increase of the number of attributes, impacts the complexity of the learned rules (more conditions). Rules are longer and this directly impacts the stopping criterion in use for addition of new rules. Indeed, RIPPER uses a stopping criterion based on Minimum Description Length (MDL) that represents the amounts of bits required to describe both the theory (in this case the model, the ruleset) and the data required to learn it. As adding features to the data affects the number of conditions in rules, MDL is impacted and learning can stop earlier. This explains why for *wine-quality-red* dataset for example the number of rules dropped from 22.0 to 14.3. To confirm this, we trained RIPPER with only the subset of features that are actually used at least once by RIPPER in the 10 folds. We achieved on average 52.0% accuracy (+4.5%) and 15.4 rules (+1 rule).

Another reason for a dataset not to benefit from the preprocessing step is that feature transformation is explicit and based on predefined operators, that are addition and substraction only in presented experiments. They are not enough to describe all datasets. For instance, metrics are improved very slightly for *abalone* dataset. Adding other operators along with the introduction of a grammar to control the validity of built features as proposed by [Ratle and Sebag, 2001] could help better describe the data while limiting computational cost and interpretability issues. *Adult* dataset for instance, had only one pair of features with dimensional compatibility. Addition and subtractions were applied to that pair, but were not used in rules. It did not benefit at all from the preprocessing step.

3.5 Conclusion

We proposed a preprocessing approach for learning more expressive rules with an automated dimensionally-aware feature generation based on mutual information. This preprocessing builds new features out of the (numerical) original attributes with a predefined set of operators. We have evaluated how added features improved resulting ruleset quality when learned with RIPPER algorithm, by comparing accuracy and number of rules with and without preprocessing.

Experiments were performed with 8 common UCI datasets. 7 datasets benefited from preprocessing in terms of conditions expressivity, higher accuracy and/or decreased number of rules (1 dataset could hardly benefit from it due to little feature dimensional compatibility). Although rule learning with RIPPER is impacted by the increased number of input features, when the number of added features is smaller than the initial number of features, this approach proved to be successful in learning fewer interpretable rules with more complex conditions and with comparable or better accuracy.

However, whereas the compatibility of this preprocessing approach with any other model can be seen as an advantage, it comes from the fact that it is a filtering method, i.e. completely independent from the learning step, and thus generated features are not guaranteed to be relevant for the learning step. An embedded approach could be the solution to this problem to ensure expressive features tailored for the learning step.

Part II

Neural-based approach

Chapter 4

Neural-based rule learning

Contents

4.1 Introduction

Most existing rule induction algorithms presume the availability of predicates used to represent the rules, naturally decoupling the predicate definition and the rule learning phases. Expressiveness is therefore highly impacted, we believe end-to-end neural-based approach can help couple them.

Different approaches have been proposed to learn rules with neural networks [Qiao et al., 2021, Beck and Fürnkranz, 2021a]. The main idea we are going to focus on in this section is a neural network architecture that models disjunctions and conjunctions thanks to well chosen AND and OR layers for binary classification problems. In opposition to sequential covering algorithms (like RIPPER for example [Cohen, 1995]), with a neural based approach multiple expressions are learned simultaneously with a global optimization strategy.

First, we introduce two architectures and associated training strategy to learn rules based on DNF and Algebraic Normal Form (ANF) expressions respectively before comparing their learning in the context of classification rule learning. On one hand, DNF expressions (disjunction of conjunctions) are similar to how humans think and are commonly used by BRMS for instance. On the other hand, ANF (exclusive disjunction of conjunctions) is a canonical form i.e. two equivalent logical expressions transform into the same ANF (also known as Zhegalkin polynomials or Reed-Muller expressions). Thus, we investigate whether the particularities of ANF and especially the XOR operator can benefit the learning of rules with neural networks.

Then, from a base rule learning architecture, we are able to learn relational predicates i.e. linear combinations of original attributes in the same training step as learning the rules [Kusters et al., 2022] and also propose an architecture for categorical data.

Also we investigate extending this architecture to answer multi-classification and multi-label classification problems.

4.2 Learning DNF - Base Model RN

We consider a supervised binary classification problem where we aim to predict a binary output label y, based on binary input data $\boldsymbol{x}$. We are provided with a training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$, where N is the number of training samples and $\mathcal{D}$ consists of N pairs of binary input data $\boldsymbol{x}^{(i)}$ and their corresponding binary labels $y^{(i)}$. A binary input vector $\boldsymbol{x}$ (i.e. $\boldsymbol{x}^{(i)}$) is represented as $\boldsymbol{x} \in \{0,1\}^D$, with D the number of binary features. A binary output label y (i.e. $y^{(i)}$) is represented as $y \in \{0,1\}$.

In this section, we want to build a network that aims at learning rules via disjunction and conjunction operations in a DNF expression. A Boolean function f is in DNF, if it is a disjunction of conjunction of binary input features $\boldsymbol{x}$.

$$f(x_1, \ldots, x_D) = \bigvee_i \left(\bigwedge_j^{k_i} L_{i,j} \right), \quad L_{i,j} \in \{x_1, \ldots, x_D\} \cup \{\neg x_1, \ldots, \neg x_D\} \tag{4.1}$$

An example of rule that we want the model to support in the context of fraud detection is provided in Equation 4.2.

```
if    ( the transaction occurs in a high-risk country
        and  the customer's account was recently created )
        or   the destination account is labeled as suspicious
then  transaction is fraudulent
```
(4.2)

The model should be trained such that given a binary input $\boldsymbol{x}$, the network produces an output prediction $\hat{y}$ such that $\hat{y}$ is a suitable approximation of the true binary label y.

4.2.1 Architecture

The model, referred to as Base Rule Network (RN), is illustrated in Figure 4.1 and defined as follows.

- Neurons are logical expressions that evaluate to 0 or 1.
  - input neurons $\boldsymbol{x}$ are binarized input features that have by definition a logical value.
  - hidden neurons $\boldsymbol{h}$ are conjunctions of the input features.
  - output neuron $\boldsymbol{y}$ is a disjunction of the (hidden) conjunctions.
- In boolean algebra, common logical operators are AND, OR and NOT. We assign to each of these operations a binary weight matrix that plays the role of a mask to filter between different nodes with regards to the logical operation considered.
  - $\boldsymbol{W}_{and}$ is the weight that filters neurons for the conjunction (AND) operation.
  - $\boldsymbol{W}_{or}$ is the weight that filters neurons for the disjunction (OR) operation.
  - $\boldsymbol{V}$ is the weight that filters neurons for the optional negation of the neurons of the conjunction (AND) operation ($v = 1$, neuron negated).

– In this model we do not allow for negation in the OR layer (as in DNF expressions).

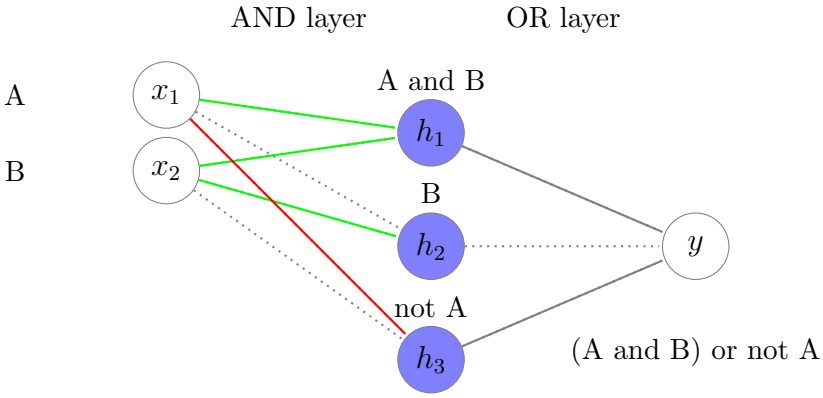


Figure 4.1: Example of trained model for rule 4.2 - Dotted (plain) lines represent masked (selected) weights. Green (red) lines represent masked (negated) weights for the NOT operation. Filled nodes are hidden nodes.

OR Layer Disjunction operation ($\vee$) is implemented by taking the minimum value between the sum of the selected elements and 1. If none of them are activated then $y = 0$, and y is equal to 1 otherwise ($\boldsymbol{y}$ is a scalar in the context of binary classification). Here, as well as in the remainder of this manuscript, $\mathbf{1}$ denotes a column vector of ones, the specific dimension of which can be inferred from the context.

$$\boldsymbol{y} = \min(\boldsymbol{W}_{or}\boldsymbol{h}, \mathbf{1}) \tag{4.3}$$

AND Layer In order to implement the conjunction operation ($\wedge$), we use the De Morgan's law that express the conjunction with the OR operator $A \wedge B = \neg(\neg A \vee \neg B)$. Applying it to Equation 4.3 we have:

$$\boldsymbol{h} = \neg(\min(\boldsymbol{W}_{and}(\neg\boldsymbol{x}), \mathbf{1})) = \mathbf{1} - \min(\boldsymbol{W}_{and}(\mathbf{1} - \boldsymbol{x}), \mathbf{1}). \tag{4.4}$$

NOT Operator It is also possible to learn the NOT operator ($\neg$) in the AND layer. To do so we can to use a binary weight $\boldsymbol{V}$ with same dimensions as $\boldsymbol{W}_{and}$ to negate input values $\boldsymbol{x}$ in the conjunctions. Modified input neurons $\hat{\boldsymbol{x}}$ represent the result of that negation if needed on $\boldsymbol{x}$ ($*$ sizes are adjusted accordingly).

$$\hat{x} = |\boldsymbol{V}^* - \boldsymbol{x}^*| \tag{4.5}$$

Equation 4.4 is modified to implement conjunction operation with possible negation.

$$\boldsymbol{h} = \mathbf{1} - \min(\boldsymbol{W}_{and}(\mathbf{1} - \hat{\boldsymbol{x}}), \mathbf{1}) \tag{4.6}$$

It is also possible to simply pad to the input $\boldsymbol{x}$ the negated input $\mathbf{1} - \boldsymbol{x}$ and let the model learn to not represent contradictions ($A \wedge \neg A$ for example). In the following, the NOT operator is not included by default in the base rule model RN unless stated otherwise.

To conclude, the base rule model is composed of a logical AND layer and an OR layer. The formal grammar that this architecture can express is specified with the following production rules (see Appendix C.1 for the full grammar):

$$\begin{aligned} \text{rule} &\rightarrow \textit{if}\ \text{base expression}\ \textit{then}\ \textit{class} = 1\ \textit{else}\ \textit{class} = 0 \\ \text{base expression} &\rightarrow \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\ \text{conjunction} &\rightarrow \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \\ \text{predicate} &\rightarrow \underline{x_1} \mid \underline{x_2} \mid \dots \end{aligned} \tag{4.7}$$

This grammar is also limited by the model architecture: *conjunction* contains at most one occurrence of each *predicate* and the total number of *conjunction*(s) is bounded by the number of hidden nodes.

It is important to note that both AND and OR operators presented here are commutative operations. In practice, in a BRMS or a programming language, the evaluation of these operations might not respect the commutativity. This has an impact in case of errors. For example for the disjunction operation, $True \vee Error$ would return an error in a commutative setting while if only the first operand is evaluated, $True$ would be returned. This is something to keep in consideration for usage of the learned rules.

Also disjunctions in natural language are often understood as exclusive disjunctions. For example when saying 'I would like this or that', 'this and that' would not be considered without the mention of 'or both' [Aloni, 2023]. Both in the model computation and in the rule examples (for example Equation 4.2), 'or' and $\vee$ refer to non exclusive disjunctions.

4.2.2 Training

In this section, we delve into the training strategy used to optimize the presented model. The overall algorithm is presented in Algorithm 3.

During training, the objective is to find the suitable model parameters Θ (i.e. $\boldsymbol{W}_{and}$, $\boldsymbol{W}_{or}$ and $\boldsymbol{V}$ when applicable) that minimize a chosen loss function $\ell(y, \hat{y})$ over the training dataset $\mathcal{D}$. This loss function quantifies the

dissimilarity between the predicted binary output $\hat{y}$ and the true binary label y.

Given a model f with parameters Θ, the machine learning problem can be mathematically formalized as the minimization of the empirical risk:

$$\min_{\Theta} \mathcal{L}(\Theta) = \min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \Theta)\right)$$

where:

- Θ represents the model parameters to be learned.
- $\mathcal{L}(\Theta)$ is the empirical risk with parameters Θ.
- $f(\boldsymbol{x}^{(i)}; \Theta)$ is the model's prediction for the binary input $\boldsymbol{x}^{(i)}$ with parameters Θ.
- $\ell(y^{(i)}, f(\boldsymbol{x}^{(i)}; \Theta))$ is the loss function that measures the dissimilarity between the true binary label $y^{(i)}$ and the predicted binary label $f(\boldsymbol{x}^{(i)}; \Theta)$.

To solve this machine learning problem, different gradient-based optimization techniques are available to find the values of the model parameters Θ that minimize the empirical risk.

In the following, we define a loss function that penalizes complex rules and specify the model parameters and optimization strategy used for the training of RN model.

To overcome training challenges attributed to binarized neural networks [Geiger and Team, 2020] (gradient zeroed almost everywhere due to the binarization operation), latent weights are used for the training of the model parameters Θ, i.e. $\boldsymbol{W}_{and}$, $\boldsymbol{W}_{or}$ and $\boldsymbol{V}$ when applicable. The model is trained via automatic differentiation (Pytorch [Paszke et al., 2019]) with Adam optimizer.

Latent weights The binary model parameters introduced above ($\boldsymbol{W}_{and}$, $\boldsymbol{W}_{or}$ and $\boldsymbol{V}$ when applicable) are trained indirectly via the training of a continuous parameter $\boldsymbol{C}$ which is activated by a sigmoid function during training and binarized with a Heaviside function during testing. With such binary weights and continuous relaxation Equations 4.3 and 4.4 are differentiable [Kusters et al., 2022]. To overcome training limitations, we use a hard concrete distribution [Qiao et al., 2021, Louizos et al., 2018]. As shown in Equation 4.9, it rescales the weights and the random variable U (uniform distribution) introduced during training, prevents from obtaining local minima. See [Louizos et al., 2018] for details.

$$\mathrm{u} \sim U(0,1), \quad \boldsymbol{S} = \sigma((log(\mathrm{u}) - log(1-\mathrm{u}) + \boldsymbol{C})/\beta), \quad \hat{\boldsymbol{S}} = \boldsymbol{S} * (\zeta - \gamma) + \gamma \tag{4.8}$$

$$\boldsymbol{W} = \min(\max(\hat{\boldsymbol{S}}, 0), 1) \tag{4.9}$$

We use the same parameter values as the original paper: $\beta = 2/3$, $\zeta = 1.1$ and $\gamma = -0.1$ [Louizos et al., 2018].

Weight values are in $[0, 1]$ during training, while for testing and rule extraction, a Heaviside is applied ($\text{heaviside}(x) = \mathbf{1}_{x \geq 0}$) to ensure strict binarization.

$$\boldsymbol{W} = \text{heaviside}(\boldsymbol{C}) \tag{4.10}$$

RN model parameters Θ ($\boldsymbol{W}_{and}$, $\boldsymbol{W}_{or}$ and $\boldsymbol{V}$ when applicable) are therefore trained indirectly and actual trained parameters are $\boldsymbol{C}_{and}$, $\boldsymbol{C}_{or}$ and $\boldsymbol{C}_{neg}$ when applicable.

Loss function We define an empirical risk $\mathcal{L}$ composed of a mean-squared error loss along with a regularization term Π that penalizes the complexity of the rule. The importance of the regularization is controlled by the parameter λ.

$$\mathcal{L}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \ell_{se}\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \Theta)\right) + \lambda \Pi(\Theta) \tag{4.11}$$

$$\ell_{se}\left(y, \hat{y}\right) = (\hat{y} - y)^2 \tag{4.12}$$

The regularization term Π, or *penalty*, evaluates the number of predicates (or terminal conditions) in the rule i.e. the number of path activated in the model as shown in Equation 4.13. In practice, we use $\lambda = 10^{-4}$.

$$\Pi(\Theta) = \Pi(\boldsymbol{W}_{and}, \boldsymbol{W}_{or}) = \boldsymbol{W}_{or}\boldsymbol{W}_{and}\mathbf{1} \tag{4.13}$$

Note that if $\boldsymbol{V}$ is applicable, i.e. if the NOT operator is included in the model, it is not taken into account in the computation of the regularization term.

It should also be emphasized that the interpretability of the approach is therefore provided by the learning of rules, but also by the optimization of the size and therefore complexity of the learned expressions via the regularization term Π.

Optimizer When using a training strategy based on latent weights, Adam optimizer is recommended because of its fast convergence and for being less sensitive to the choice of hyperparameters in this context [Geiger and Team, 2020]. However, it is also possible to train a BNN in a latent free manner with a custom optimizer.

Bop is a custom optimizer for BNN that only has one possible action : flipping weights by taking into account the consistency and strength of the gradients [Helwegen et al., 2019]. We tried training the base rule model with Bop. It required adapting Bop to boolean weight problems as described in Equation 4.14. We kept the notation from the original paper with m_t^i the exponential moving average at time t of weight w_t^i (with adaptability rate γ for consistency) and τ a threshold to control the strength of the gradient required to flip. Exponential moving average computation is kept unchanged.

$$w_t^i = \begin{cases} 1 - w_t^i & \text{if } |m_t^i| \geq \tau \text{ and } (m_t^i \geq 0) = w_t^i \\ w_t^i & \text{otherwise} \end{cases} \tag{4.14}$$

However in practice we were not able to obtain close to comparable results with Bop optimizer. We tried multiple combinations of parameters, network size and datasets but we found the configuration to be very tricky to configure and were not able to find a stable training strategy. We want to stress that we are not saying this optimizer can not work (and it does in some cases) but we were not able to find an ideal setting on noise free simple synthetic datasets. One observation we were able to make is that in the rare settings models did converge to an accurate trained model, the complexity of the rules was high. Also in the context of the original paper the optimizer is tested on models that are not fully binary (most of the layers are but not all) this might have an impact on the strength of the gradients and the stability of this approach.

Due to the fact that training strategy with latent weights worked, we did not delve further into this strategy but this direction might be of interest for future work.

4.3 Learning ANF - Model RN-anf

DNF are very natural expressions to learn as they are easily comprehensible by humans [Fürnkranz et al., 2012], and are used in BRMS for instance. However, any boolean function can be expressed in other normal forms such as ANF (exclusive disjunction of conjunctions) a canonical form also known as Zhegalkin polynomials or Reed-Muller expressions. In this section, we investigate whether the particularities of ANF and especially the XOR operator

Algorithm 3: Training procedure based on mini-batch training

Data: Training dataset $\mathcal{D}$, Learning rate: η, Batch size: B, Number of epochs: E
Result: Trained neural network model
Initialize neural network parameters;
for $i \leftarrow 1$ **to** E **do**
  for $j \leftarrow 1$ **to** $\frac{N}{B}$ **do**
    Randomly sample a mini-batch $\mathcal{D}_{\text{batch}}$ of size B from $\mathcal{D}$;
    Forward pass: Compute predictions on mini-batch using the current parameters (Eq. 4.3, 4.4 or 4.6);
    Compute the loss between predictions and actual targets (Eq. 4.11);
    Backpropagation: Compute gradients of the loss with respect to the parameters;
    Update the parameters using Adam optimizer with learning rate η;
return *Trained model*;

($\veebar$) can benefit the learning of rules with neural networks.

The setting and objective from the previous Section 4.2 are still valid. However in this section, we want to build a network that aims at learning rules via exclusive disjunction and conjunction operations in an ANF expression. A Boolean function f can be uniquely represented in a ANF.

$$f(x_1, \ldots, x_D) = \bigvee_{j \in \overline{\{0,1\}^D}} a_j \left(x_1^{j_1} \wedge \cdots \wedge x_D^{j_D} \right) \tag{4.15}$$

where $j = \{j_1, \ldots, j_D\} \in \{0,1\}^D$ and coefficients a_j take values in $\{0,1\}$.

Any rules that could be learned in Section 4.2 can be converted to ANF. Example from Equation 4.2 is rewritten in ANF in Equation 4.16 with variable names rather than text expressions for readability.

$$\begin{aligned} &\texttt{if} \quad \text{isDestAccountSusp} \veebar (\text{isHighRiskCountry} \wedge \text{isNewAccount}) \\ &\qquad \veebar \quad (\text{isDestAccountSusp} \wedge \text{isHighRiskCountry} \wedge \text{isNewAccount}) \\ &\texttt{then} \quad \text{transaction is fraudulent} \end{aligned} \tag{4.16}$$

4.3.1 Architecture

The ANF rule model invoked is composed of two consecutive layers: an AND layer (Section 4.2.1) and a XOR layer described below. A fixed True value is also added as input to the XOR layer to match the ANF grammar. Negations are not allowed in ANF but this True term negates the result of the rest of the XOR expression due to parity properties. The model, referred to as the ANF Rule Network (RN-anf), is illustrated in Figure 4.2.

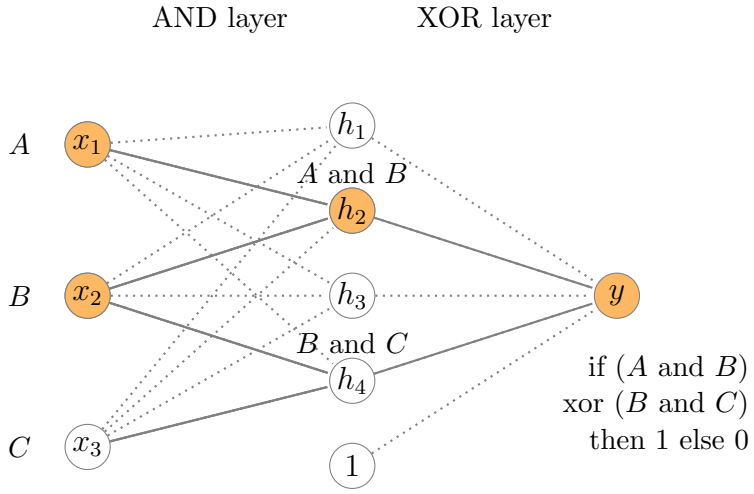


Figure 4.2: Example of a trained RN-anf model for the rule *if (A and B) xor (B and C) then 1 else 0* on binary predicates A, B and C . Plain (dotted) lines represent activated (masked) weights. An example evaluation of the model is represented with the filled neurons (neuron=1) for the binary input $A = 1$ and $B = 1$.

XOR Layer Exclusive disjunction operation ($\veebar$) can be interpreted in terms of parity as the result is True if the number of true inputs is odd, and False if the number of true inputs is even. We exploit this property to implement the XOR layer using cosine function, with $\boldsymbol{W}_{xor}$ associated binary weights as follows.

$$y = \frac{1}{2}\left(1 - \cos\left(\pi \boldsymbol{W}_{xor} \begin{bmatrix} \boldsymbol{h} \\ 1 \end{bmatrix}\right)\right) \tag{4.17}$$

Although this function is continuous, it is important to note that when the input values and/or weights are in between 0 and 1 the result can be completely meaningless.

The formal grammar of the expressions that can be expressed with this model is specified with the following production rules (see Appendix C.2 for the full grammar):

$$\begin{aligned} \text{base expression} &\rightarrow \text{conjunction} \mid \text{conjunction} \veebar \text{base expression} \\ \text{conjunction} &\rightarrow \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \mid \textit{True} \\ \text{predicate} &\rightarrow \underline{x_1} \mid \underline{x_2} \mid \dots \end{aligned} \tag{4.18}$$

It is important to note that, as shown in the grammar, this architecture does not prevent from having duplicate conjunctions in the exclusive disjunction. Thus we are learning ANF-like expressions that can be very easily simplified as a post processing step if needed. We are not applying this post processing of the rules in the following unless clearly stated. Equation 4.15 is modified into the following to describe an ANF-like expression.

$$f(x_1, \dots, x_D) = a \veebar \bigvee_i \left(\bigwedge_j^{k_i} L_{i,j} \right), \quad L_{i,j} \in \{x_1, \dots, x_D\}, \quad a \in \{0, 1\} \tag{4.19}$$

4.3.2 Training

In the same manner as in Section 4.2.1, in order to overcome the model to be binary, we use latent weights (Section 4.2.2). All operations of the previously described model are differentiable. Model parameters Θ, i.e. $\boldsymbol{W}_{and}$ and $\boldsymbol{W}_{xor}$, are trained through automatic differentiation. For model training, we use mean-square error loss (MSE) function and Adam optimizer (learning rate $\eta = 0.1$).

Loss function Loss function from Equation 4.11 is still valid. In practice, we use $\lambda = 10^{-4}$. The number of terminal conditions of the model Π can be expressed as follows.

$$\Pi(\Theta) = \Pi(\boldsymbol{W}_{and}, \boldsymbol{W}_{xor}) = \boldsymbol{W}_{xor} \begin{bmatrix} \boldsymbol{W}_{and}\mathbf{1} \\ 1 \end{bmatrix} \tag{4.20}$$

It is important to note that although rules might require to be converted back to DNF and thus be expressed in more or less terminal conditions, we are minimizing the complexity of the ANF expression here. This is a limitation of this approach as converting back and forth between ANF and DNF especially for simplified versions of the DNF can be very costly but also non differentiable and thus not applicable for the computation of the penalty during training.

#	**Ground Truth**	**+class**	π	π^*_{DNF}	π^*_{ANF}
1	$F_0 \vee (F_3 \wedge F_6)$	63.5	3	3	6
2	$(F_0 \wedge F_4) \vee (F_3 \wedge F_6)$	43.6	4	4	8
3	$(\neg F_3 \wedge \neg F_6) \vee \neg F_0 \vee \neg F_4$	80.6	4	4	11
4	$(\neg F_3 \wedge \neg F_6) \vee (F_0 \wedge F_9 \wedge \neg F_6) \vee \neg F_0 \vee \neg F_4$	83.9	7	6	20
5	$F_0 \veebar F_4$	49.6	2	4	2
6	$F_0 \veebar F_4 \veebar F_8$	49.9	3	12	3
7	$F_0 \veebar F_4 \veebar (F_3 \wedge F_8)$	49.5	4	20	4
8	$F_0 \veebar F_4 \veebar (F_8 \wedge \neg F_3)$	50.2	4	20	5

Table 4.1: Ground truths applied on binary input features $F_0, \ldots, F_9 \in \{0,1\}^{10}$ to generate the 8 synthetic datasets along with the proportion of the positive class (in %), π the number of conditions in the expression, π^*_{DNF} the number of conditions in the simplified equivalent DNF expression and π^*_{ANF} the number of conditions in the equivalent ANF expression.

4.4 Comparing the learning of DNF and ANF

In order to compare the learning of DNF to ANF expressions in the context of neural-based rule learning model, we apply different models to synthetic datasets and compare them in terms of training performance and training time. Two different DNF rule learning model implementations and one ANF rule learning model are compared.

4.4.1 Experiments

The first DNF model implementation corresponds to the AND layer with the NOT operator followed by the OR layer introduced in Section 4.2.1. It will be refereed to as the AND-NOT-OR model. The second DNF model implementation corresponds to an AND layer followed by an OR layer introduced in Section 4.2.1 with both input values and negated input values provided as input to the model. It will be refereed to as the AND-NOTF-OR model. The ANF model is RN-anf composed of AND layer followed by a XOR layer.

Synthetic datasets We propose 8 synthetic datasets for discovering simple binary classification rules composed of disjunctions, exclusive disjunctions and conjunctions as shown in Table 5.1. These are composed of 10000 instances with 10 binary features. Generation is detailed in Appendix A.

Experimental Setting All datasets are partitioned in a stratified fashion with 60% for training, 20% for validation and 20% for testing datasets and we use a batch size of 100 instances and train for 200 epochs. The hidden size in the model is set to 60 which is at least three times the maximum of π^*_{DNF} and π^*_{ANF}. More details on experimental setting can be found in Appendix E. For each experiment, we run the algorithm 10 times with different random seeds for weights initializations. Resulting metrics are averaged over these runs.

4.4.2 Results and Discussion

	AND-XOR		**AND-NOT-OR**		**AND-NOTF-OR**	
#	**Accuracy**	**Penalty**	**Accuracy**	**Penalty**	**Accuracy**	**Penalty**
1	100.0 ± 0.0	8.8 ± 1.9	100.0 ± 0.0	**6.4 ± 0.5**	100.0 ± 0.0	7.2 ± 0.6
2	100.0 ± 0.0	8.0 ± 0.0	100.0 ± 0.0	**7.4 ± 0.5**	100.0 ± 0.0	8.6 ± 0.3
3	**99.7 ± 0.3**	11.2 ± 0.2	100.0 ± 0.0	**7.4 ± 0.6**	100.0 ± 0.0	9.8 ± 0.5
4	**96.8 ± 0.0**	5.5 ± 1.5	100.0 ± 0.0	9.9 ± 0.6	100.0 ± 0.0	11.4 ± 0.7
5	**88.0 ± 6.5**	41.6 ± 10.3	100.0 ± 0.0	6.4 ± 0.4	100.0 ± 0.0	8.4 ± 0.5
6	**86.2 ± 6.6**	60.5 ± 12.8	100.0 ± 0.0	14.1 ± 0.9	100.0 ± 0.0	15.0 ± 1.1
7	100.0 ± 0.0	26.0 ± 8.0	96.3 ± 1.9	16.7 ± 2.2	**97.6 ± 1.0**	21.6 ± 1.4
8	**89.8 ± 6.8**	50.9 ± 11.2	98.1 ± 0.9	19.6 ± 0.7	**99.3 ± 0.7**	21.4 ± 1.5

Table 4.2: Average accuracies (%) and penalties Π obtained for the different datasets, along with the unbiased standard errors of the average over the 10 executions with different weights initializations.

ANF vs DNF In terms of accuracy, average accuracies over the 10 executions in Table 4.2 highlights that DNF models overall outperform the ANF model. Figure 4.3 consolidates this observation and highlights the high standard error over the 10 runs for the ANF model. The training of ANF model is not as effective as for other two DNF models. What happens during training in terms of training loss is shown in Figure 4.4. DNF models converge quickly (less than 20 epochs) and learn in all cases, while the ANF model does not. Also, minimum ANF training loss is almost always higher than maximum DNF models training loss. Except for dataset 4, which is the dataset with the highest π^*_{ANF} value, ANF model converged at least once for all the other datasets towards the ground truth.

All these observations lead us to the conclusion that this ANF model is not as effective in terms of training as the presented DNF models. There are two main reasons, we can think of to explain this.

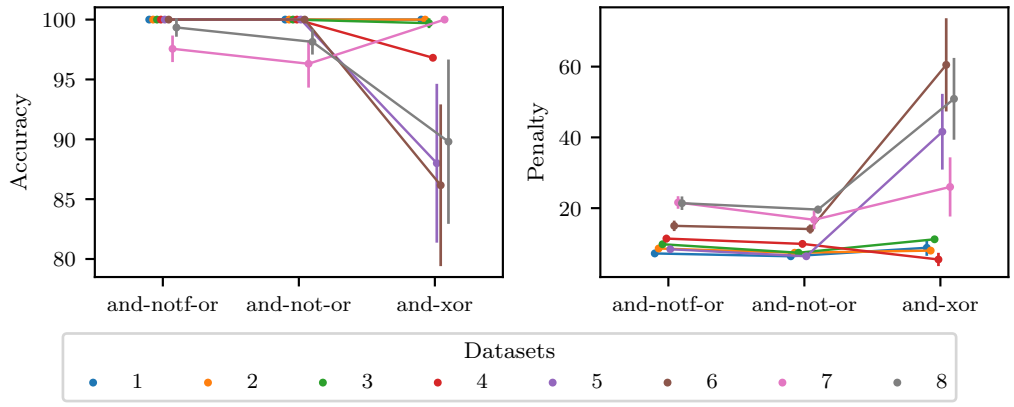


Figure 4.3: Average accuracies (%) and penalties Π obtained for the different datasets per model, along with the standard errors of the average over the 10 executions with different weights initializations.

First, for a fixed hidden size, as there are less possible local optima in search space due to the properties of ANF, training is more sensitive to weights initialization and ideal training hyper parameters are likely to be different.

Then, both neurons and weights when training starts are in between 0 and 1, meaning that the resulting XOR computation is completely meaningless (Equation 4.17). Gradients are therefore significantly impacted. AND and OR operations are also not very meaningful when neurons and weights are in between 0 and 1. This is the case when $\boldsymbol{W}_{or}\boldsymbol{h}$ in Equation 4.3 is higher than $\mathbf{1}$ for the OR operator for example, as Equation 4.3 no longer approximates a logical OR. However, in practice and thanks to the regularization term in the loss function, the first training steps start by zeroing a lot of weight values (or at least for Π to be small if not zero) before converging towards a minima. This model simplification is beneficial for the AND and OR logical operations approximations and overall penalty of the final expression. The XOR computation however does not benefit from it : it only provides meaningful approximation when only at most one term value of the weighted sum $\boldsymbol{W}_{xor}\begin{bmatrix}\boldsymbol{h}\\1\end{bmatrix}$ from Equation 4.17 is in between 0 and 1. It also explains why the ANF model performs better on datasets where ground truths contain conjunctions. Also on these very simple synthetic data, DNF models are not impacted by approximation errors likely because of the simplicity of the ground truths.

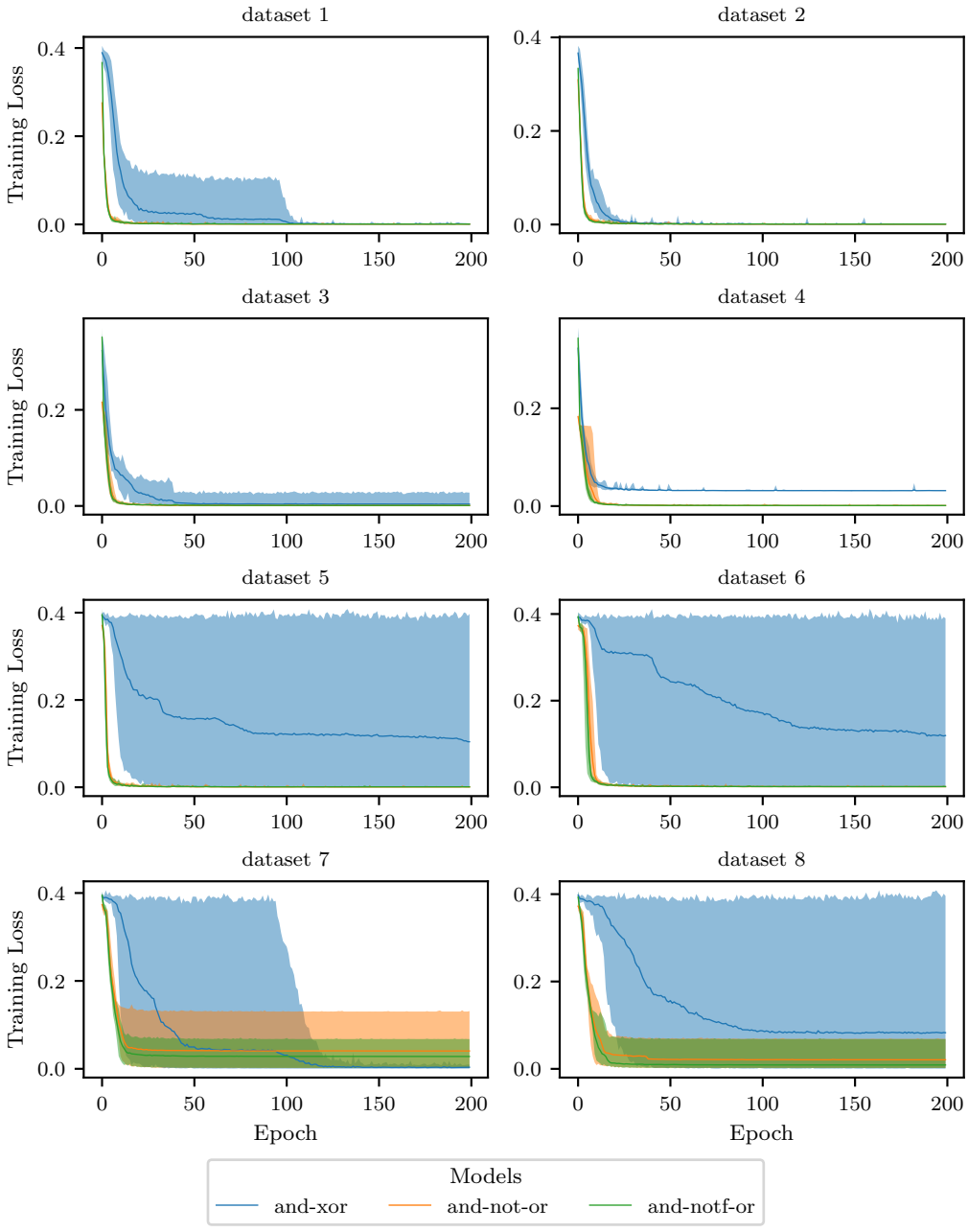


Figure 4.4: Average, min and max training loss of the different models on all synthetic datasets during training. Area between the min and max training loss values is filled in the corresponding color for readability.

DNF implementations Both DNF implementations have comparable training loss variations (Figure 4.4) and differ only slightly by a marginally higher

Model	Time per epoch (ms)
and-xor	4.54 ± 0.06
and-not-or	12.33 ± 0.18
and-notf-or	4.20 ± 0.05

Table 4.3: Average time per epoch measured for all experiments per model along with standard error of the average.

accuracy for one (and-notf-or) and a marginally lower penalty for the other (and-not-or). This is intuitively understandable given their difference in implementation as and-notf-or maximum theoretical penalty is twice bigger than and-not-or model and and-not-or model has an additional step (Equation 4.5) impacting training. Their biggest difference is in the average execution time of an epoch with model and-not-or being 3 times slower. This is probably because the operations required to calculate a selective negation of inputs are more time-consuming than a negation "in all cases". Note that the quality of the implementation could also affect the performances of the models.

4.4.3 Conclusion

In conclusion, we have compared the learning with a neural-based approach of an ANF expression to a DNF expression and evaluated their training performances in a simplistic context. The training of the ANF rule model turned out to be less stable without providing any other gains in performance. We have not found sufficient motivation to pursue the direction of learning ANF. Therefore, in the following we will continue focusing on the base rule model presented in Section 4.2.1 that has demonstrated more favorable outcomes in our study.

As part of this study we also compared two implementations of DNF rule model, the difference lying in the way they handle negations. Although more experiments would be needed to assess when one of the approaches should be preferred (for example, depending on the input size of the model), it emerged that the learning time is multiplied by 3 in our case when learning negation via a weight (and-not-or) compared with padding the input with the negated values (and-notf-or).

4.5 Data types

The base model architecture (RN) presented in Section 4.2.1 supports binary input data (input values passed to the model are boolean). In the same manner as for state-of-the-art rule learning algorithms like RIPPER [Cohen, 1995] for example, a separate preprocessing step can be applied on the data to binarize the data. However, by opting for an end-to-end approach, the predicates can be learned alongside the rules themselves.

In the following, we extend the architecture to support numerical and categorical data and thus augment the expressivity of the end-to-end approach. Other data types like sequences will be studied in Chapter 5.

4.5.1 Numerical Data

Modified version of literal learning module from [Kusters et al., 2022].

By learning rules from numerical data, we imply that we learn a binarization of the numerical attributes from which we learn rules. An example of binarized expression in the context of fraud detection is provided in Equation 4.21.

$$\text{the amount of the transaction is higher than } 10000 \tag{4.21}$$

Numerical data $\boldsymbol{x} \in \mathbb{R}^D$ can be transformed into boolean input data with a sigmoid activation (binarization) to be provided as input to the base model RN.

With a simple linear layer with real-valued trainable weight $\boldsymbol{W}$, bias $\boldsymbol{b}$ and a sigmoid activation (a.k.a perceptron), output node $\boldsymbol{\phi}_j$ represents a numeric comparison on a linear combination of the input features $\sum w_{ji}x_i \geq b_j$.

$$\boldsymbol{\phi} = \sigma\left(\frac{\boldsymbol{W}\boldsymbol{x} - \boldsymbol{b}}{\theta}\right) \tag{4.22}$$

σ is the sigmoid activation function applied component-wise with temperature θ which controls the slope of the sigmoid.

For validation, testing and rule extraction, the sigmoid is replaced by a Heaviside step function to ensure strict binarization. It is a choice of implementation to fix $\boldsymbol{\phi} = 0$ or $\boldsymbol{\phi} = 1$ when Heaviside input is 0 to represent $\sum w_{ji}x_i > b_j$ or $\sum w_{ji}x_i \geq b_j$ expressions respectively.

This layer that we will refer to as the *Linear Combination layer*, can also be constrained to represent direct numeric comparisons of individual input features by fixing the weights matrix W to be the identity matrix. This constrained layer will be referred to in the following as the *Threshold layer*.

It is worth highlighting that a Threshold layer is able to represent $x_i > b_j$ but can not represent $x_i \leq b_j$ as all the non-null weights are positive. Both the Linear Combination layer and the Threshold layer can be input layers to the base model RN for numerical data processing and will be referred to as Numerical layers.

Also if additional information were to be available on the nature of the input data such as their dimensions for instance, Linear Combination layers could be constrained to match with the associated requirements in that case by ensuring the representation of dimensionnally-compatible features.

In terms of training, it is important to note that input data of numerical layers should be standardized either from scratch or via batch normalization especially when combined with initialization distributions for weights and bias that are centered around zero. When extracting the rules, standardization or batch normalization operations can be reversed to rescale the weights and bias values for usage in the rules. We opt in this work for Xavier uniform initialization of weights and bias [Glorot and Bengio, 2010] and batch normalization applied before the Numerical layers.

The formal grammar that this architecture can express is specified with the following production rules:

$$\begin{aligned} \text{linear combination literal} &\rightarrow \underline{\sum w_{ji} x_i} \textit{ is greater than } \underline{b_j} \\ \text{threshold literal} &\rightarrow \underline{x_i} \textit{ is greater than } \underline{b_j} \end{aligned} \tag{4.23}$$

4.5.2 Categorical Data

Introduced in [Collery et al., 2023].

By learning rules from categorical data, we imply that we learn rules from a binary representation or evaluation of the categorical attributes. An example of evaluation of categorical expression in the context of fraud detection is provided in Equation 4.24.

$$\text{the id of the destination account of the transaction is “UniqueID”} \tag{4.24}$$

Categorical data can easily be transformed into binary input data with one-hot encoding to be provided as input to the base model RN. However in theory impossible logical expression can be represented with this strategy.

Let x_c be a categorical feature that can take one value α_i^c out of a fixed number of possible values n. Binary inputs $x_c = \alpha_0^c$ and $x_c = \alpha_1^c$ are then given as input to the AND layer (as shown by production rules in Equation 4.25) that can in theory represent the impossible expression

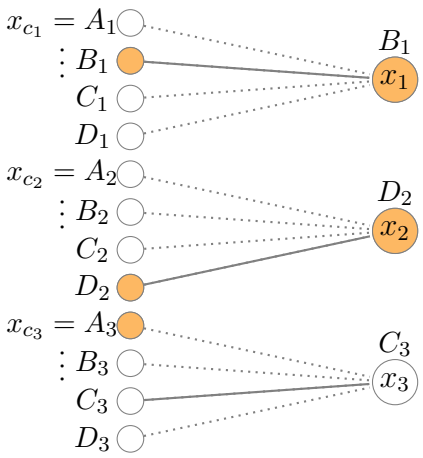


Figure 4.5: Example of a trained StackedOR layer on 3 categorical variables x_{c_1}, x_{c_2} and x_{c_3} ($x_{c_k} \in \{A_k, B_k, C_k, D_k\}$). For simplicity, the truth value of $x_{c_1} = B_1$ is replaced by B_1 for example. Plain (dotted) lines represent activated (masked) weights. An example evaluation of the model is represented with the filled neurons (neuron=1) for the binary input $x_{c_1} = B_1$, $x_{c_2} = D_2$ and $x_{c_3} = A_3$.

$x_c = \alpha_0^c \wedge x_c = \alpha_1^c$ i.e the model has to learn the hidden categorical relationship between the one-hot encoded variables. To prevent learning it when we already know, we propose a stacked architecture of OR layers for categorical data, referred to as the StackedOR layer, that can be plugged in the base model RN as shown in Figure 4.5. Assuming K categorical features, this structure is defined by K weights, $\boldsymbol{W}_{stack}^k$, (one for each categorical input) that are combined in a block-diagonal weight matrix $\boldsymbol{W}_{stack}$. $\boldsymbol{W}_{stack}$ binary weights are trained in the same manner as $\boldsymbol{W}_{and}$ and $\boldsymbol{W}_{or}$ as latent weights (Section 4.2.2).

$$\text{categorical literal} \rightarrow \underline{x_c = \alpha_0^c} \mid \ldots \mid \underline{x_c = \alpha_n^c} \text{ (or simply } \underline{\alpha_0^c} \mid \ldots \mid \underline{\alpha_n^c}) \tag{4.25}$$

However, with this layer in front of the AND layer, the learned expression is constrained to only have one set of possible values for each x_c in the final expression, i.e. it prevents from learning multiple disjunctions of x_c values to be used in different rule disjunctions. This can be fixed by increasing the output size of the stacked OR layers and masking in the following AND layers impossible combinations.

4.6 Multi-classification and Multi-label Classification Problems

Multi-classification With any binary classification model and a one-against-all strategy, a binary classification model can be used for multi-classification problems. This strategy to deal with multiple classes can be adapted to the learning of ordered rules as presented for RIPPER [Cohen, 1995] but it can be applied to any model. It consists in learning an ordered set of rules by ordering the n classes by increasing prevalence and training the model on the binary classification problems of class i against classes $\{i+1, \ldots, n\}$ for all values of $i \in \{1, \ldots, n-1\}$, last class (class n) being the default class (else clause).

This approach could be applied to our base architecture (RN) but a thorough examination and study of the model robustness on unbalanced datasets would be required to ensure qualitative and quantitative results. Indeed, this is pointing out a common tradeoff for rule learning : quality of the learned rules (measured for example with accuracy) and their simplicity or length (measured for example with the number of conditions in the rules). In an unbalanced environment, and especially with a gradient-based optimization, learning the empty rules if TRUE then ... or if FALSE then ... will likely be highly favorable local minima without more precautions taken during training or with the data.

RL-Net, another approach for multi-classification was recently introduced with the learning of ordered rules [Dierckx et al., 2023]. It consists in adding a hierarchy layer to force an order between the neurons (or rules) that is followed by an output layer to assign those rules to a class. This strategy would be our preferred choice for extending the model for multi-classification.

Multi-label classification Another type of problem the base architecture can be applied to is multi-label classification. It consists of labeling instances with non-exclusive classes. The architecture can be adapted to multi-label classification problems by having an output node per non-exclusive class. A disjunction of conjunction will represent each class. However this solution is a first step to tackle multi-label classification as without any additional changes, the model would not capture dependencies between the different labels.

4.7 Conclusion

To conclude, in this chapter we have presented a base neural architecture to learn binary classification rules along with an associated training strategy and explored to some extent different directions of improvement and adaptation to address a wider range of problems. With a neural based approach multiple expressions are learned simultaneously with a global optimization strategy and the expected complexity (or length) of the rules can be monitored by the regularization coefficient in the loss function.

After trying out an unsuccessful latent-free training strategy, we confirmed our working strategy for the future experiments with latent weights.

We studied whether the properties of ANF and especially the XOR operator could benefit the learning of rules with neural networks. In practice, on simplistic scenarios we found the training to be less stable than when training a DNF without gain in performance or training time.

We also presented methods for dealing with numerical and categorical data and discussed how the base neural architecture could be extended to other common supervised machine learning problems (multi-classification and multi-label classification).

This chapter has given us an insight as to the flexibility of this approach and reinforces our view that the base rule model (RN) can be used as a basis for more complex architectures, i.e. models that can learn rules governed by more complex (and expressive) grammars.

Chapter 5

Neural-based rule learning for sequential data

Contents

5.1 Introduction

Discovering interpretable patterns for classification of sequential data is of key importance for a variety of fields, ranging from genomics to fraud detection or more generally interpretable decision-making. We believe in our end-to-end neural-based approach to be of great interest also for sequential data. Both because of its compatibility with the different input blocks presented previously (learning of literals from different data types in an end-to-end manner) and because of the possible compatibility of the resulting learned sequential patterns with other layers.

In this chapter, we investigate two approaches to learn rules from sequential data based on the base architecture. The first one is a Recurrent Neural Network (RNN) and the second a Convolutional Neural Network (CNN).

In this chapter, we consider a supervised binary classification problem where we aim to predict a binary output label y, based on multivariate sequential input data $\boldsymbol{X}$. We are provided with a training dataset $\mathcal{D} = \{(\boldsymbol{X}^{(1)}, y^{(1)}), (\boldsymbol{X}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{X}^{(N)}, y^{(N)})\}$, where N is the number of training samples and $\mathcal{D}$ consists of N pairs of multivariate sequential data $\boldsymbol{X}^{(i)}$ and their corresponding binary labels $y^{(i)}$. A multivariate sequence $\boldsymbol{X}$ (i.e. $\boldsymbol{X}^{(i)}$) of D features and of length T is defined as $\boldsymbol{X} = (\boldsymbol{x}_0, ..., \boldsymbol{x}_{T-1})^T \in \{0,1\}^{T \times D}$, where $\boldsymbol{x}_t \in \{0,1\}^D$, $\forall t \in \{0, ..., T-1\}$, is the t^{th} observation of all D variables. A binary output label y (i.e. $y^{(i)}$) is represented as $y \in \{0, 1\}$.

We also introduce $\mathtt{t}$, the position when the last observation in a sequence was made. With $\mathtt{t}$ being our reference, $\mathtt{t} - i$ refers to the moment of the i^{th} observation before $\mathtt{t}$ with $i \in \{0, ..., T-1\}$).

The models presented can also be extended to a set of N sequences of $\boldsymbol{x}$ of a fixed number of feature D and of different lengths T_n.

5.2 Recurrent Rule Neural Network (RR2N)

We investigate the use of an interpretable many-to-one RNN for binary classification of sequential data.

The purpose of the network is to learn logical rules composed of disjunctions of conjunctions over sequential data. An example is provided in Equation 5.1.

$$\text{if } (A \text{ and } B \text{ at } \mathtt{t}) \text{ or } (B \text{ at } \mathtt{t}-1) \text{ then } class = y \tag{5.1}$$

where A and B are binary predicates.

A more complex example of logical rule over sequential data in the context of fraud detection is provided in Equation 5.2.

$$\begin{array}{ll} \texttt{if} & \text{the balance of the oriAccount at } \texttt{t-1} > 10000 \\ \quad \texttt{and} & \text{the balance of the oriAccount at } \texttt{t} <= 1 \\ \texttt{then} & \text{transaction is fraudulent} \end{array} \tag{5.2}$$

The learned expressions are relative to the position `t`. We arbitrary right align the sequences and thus learn patterns relative to the last observation `t` in the sequence. The alignment is an implementation detail that should be chosen according to the problem to solve.

5.2.1 Architecture

In order to learn rules that are sequence dependent, a natural extension is to add an explicit recursion to the base model RN presented in Chapter 4. This recursion is added on the AND layer while the OR layer remains unchanged. We list here the key changes made to the RN model and illustrates the new model, Recurrent Rule Neural Network (RR2N), in Figure 5.1 for Equation 5.1.

- Neurons are logical expressions that can capture a sequential dependency.
  - input neurons $\boldsymbol{x}_t$ are sequentially-dependent binarized input features.
  - hidden neurons $\boldsymbol{h}_t$ are conjunctions of the input features $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$, which is the hidden state that captures the conjunctions up to $\texttt{t} - 1$.
  - input neurons of the AND layer $\boldsymbol{x}_{and,t}$ are now $\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix}$
- The same changes are defined for the weights associated to logical operations AND and OR.
  - $\boldsymbol{W}_{and}$ is still the weight that filters neurons for the conjunction (AND) operation. However it is composed of two weights $\boldsymbol{W}$ and $\boldsymbol{U}$ that filters neurons for the AND operation on the input and hidden neurons respectively so that $\boldsymbol{W}_{and} = \begin{bmatrix} \boldsymbol{W} & \boldsymbol{U} \end{bmatrix}$.
  - $\boldsymbol{W}_{or}$ is kept unchanged.

AND Layer By replacing the new terms in Equation 4.4, and setting an initial state for the hidden nodes to 1 we have the following.

$$\begin{cases} \boldsymbol{h}_0 = \mathbf{1} \\ \boldsymbol{h}_t = \mathbf{1} - \min(\boldsymbol{W}_{and}(\mathbf{1} - \boldsymbol{x}_{and,t}), \mathbf{1}) \end{cases} \tag{5.3}$$

$$\begin{cases} \boldsymbol{h}_0 = \mathbf{1} \\ \boldsymbol{h}_t = \mathbf{1} - \min(\boldsymbol{W}(\mathbf{1} - \boldsymbol{x}_t) + \boldsymbol{U}(\mathbf{1} - \boldsymbol{h}_{t-1}), \mathbf{1}) \end{cases} \tag{5.4}$$

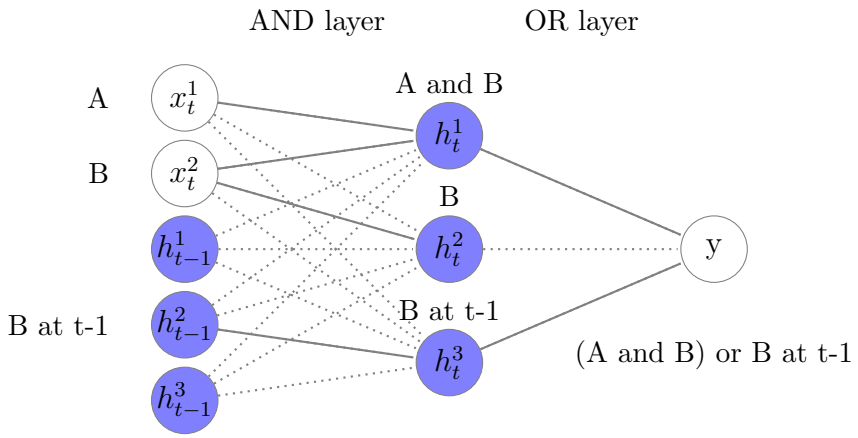


Figure 5.1: Example of trained RR2N model - Plain (dotted) lines represent activated (masked) weights. Filled nodes are hidden nodes. Equivalent exploded views of this example of RR2N model are shown in Appendix F.

The formal grammar that this architecture can express is specified with the following production rules (see Appendix C.3 for the full grammar):

$$\begin{aligned} \text{base expression} &\rightarrow \text{conjunction}_t \mid \text{conjunction}_t \vee \text{base expression} \\ \text{conjunction}_t &\rightarrow \text{literal}_t \mid \text{conjunction}_{t-1} \mid \text{literal}_t \wedge \text{conjunction}_t \\ &\quad \mid \text{conjunction}_{t-1} \wedge \text{conjunction}_t \\ \text{literal}_t &\rightarrow \underline{x^d} \; at \; \texttt{t}{-}\underline{t} \end{aligned} \tag{5.5}$$

What this grammar points out is that the learned rules are nested conjunctions each of which applies to a specific position t. Therefore rule extraction is done in a recursive manner as follows. When a node $\boldsymbol{h}_t^i$ in the hidden layer is connected to a node that is not an input node but a hidden node $\boldsymbol{h}_{t-1}^j$, time is incremented by one and and $\boldsymbol{h}_t^j$ connections are explored recursively until we reach initialization depth or until there is no hidden nodes connected to the studied node.

The hidden nodes in the AND layer are initialized to $\mathbf{1}$. This value is not meaningless as it is equivalent of a $True$ logical value. The recursive extraction should then stop at a depth of search T which is the trained sequence length with the initialization value. When extracting rules from the model and reaching a hidden node $\boldsymbol{h}^i$ at $t-(T-1)$ the value of that node is then $True$. This can easily be extended for sequences of different lengths T_n by padding and fixing $T = \max(T_n)$.

5.2.2 Training

The training strategy from Chapter 4 is kept unchanged, with model parameters Θ (i.e. $\boldsymbol{W}_{and}$ and $\boldsymbol{W}_{or}$) being trained indirectly with latent weights and automatic differentiation (see Section 4.2.2).

Loss function Loss function from Equation 4.11 is still valid. In practice, we also use $\lambda = 10^{-4}$. The number of terminal conditions of the recurrent rule model Π is expressed as follows.

$$\Pi(\Theta) = \Pi(\boldsymbol{W}_{and}, \boldsymbol{W}_{or}) = \boldsymbol{W}_{or}\boldsymbol{\Pi}_{T-1}, \quad \begin{cases} \boldsymbol{\Pi}_0 = \boldsymbol{W}_{and}\mathbf{1} \\ \boldsymbol{\Pi}_t = \boldsymbol{W}_{and}\begin{bmatrix} \mathbf{1} \\ \boldsymbol{\Pi}_{t-1} \end{bmatrix} \end{cases} \tag{5.6}$$

It is important to note that this expression does not take into account easy simplification of the learned rules and considers the initial state of True nodes as end conditions.

5.2.3 Limitations

Adding this recurrence in the model architecture brings the same limitations as a basic RNN [Bengio et al., 1994]. We are mainly referring to the problem of the vanishing gradients that we will not try to solve here, but due to that specific limitation we will focus on learning rules from sequences of short length (6 observations).

5.2.4 Experiments

Synthetic datasets We propose 6 synthetic datasets for discovering simple binary classification rules on sequential data as shown in Table 5.1. The last 4 datasets are to evaluate the magnitude of the vanishing gradient problem. These are composed of 10000 sequences of length 7 with 10 binary features. Generation is detailed in Appendix A.

Experimental Setting All datasets are partitioned in a stratified fashion with 60% for training, 20% for validation and 20% for testing datasets and we use a batch size of 100 sequences and train for 200 epochs. The hidden size in the model is set to 10. More details on experimental setting can be found in Appendix E. For each experiment, we run the algorithm 10 times with different weight initializations. Resulting metrics are averaged over these runs.

#	Ground Truth	+class	π
1	F_1 at t-1 $\wedge$ F_2 at t	25.0	2
2	F_1 at t-1 $\vee$ F_2 at t	75.1	2
3	F_4 at t-1	50.5	1
4	F_4 at t-2	50.0	1
5	F_4 at t-3	49.7	1
6	F_4 at t-4	49.6	1
7	F_4 at t-5	50.0	1

Table 5.1: Ground truths applied on binary input features $F_0, \ldots, F_9 \in \{0,1\}^{10}$ to generate 6 synthetic datasets along with the proportion of the positive class (in %) and the penalty π of the expression.

5.2.5 Results and Discussion

	200 epochs			600 epochs		
Dataset	**Accuracy**	**Bal. Acc.**	Π	**Accuracy**	**Bal. Acc.**	Π
1	100.0 ± 0.0	100.0 ± 0.0	2 ± 1	-	-	-
2	100.0 ± 0.0	100.0 ± 0.0	3 ± 1	-	-	-
3	100.0 ± 0.0	100.0 ± 0.0	1 ± 1	-	-	-
4	95.2 ± 15.2	95.0 ± 15.8	2 ± 1	94.8 ± 16.3	95.0 ± 15.8	5 ± 9
5	64.7 ± 24.4	65.0 ± 24.2	2 ± 2	90.1 ± 20.8	90.0 ± 21.1	3 ± 3
6	75.9 ± 25.6	76.1 ± 25.4	4 ± 3	74.6 ± 26.8	75.0 ± 26.4	2 ± 2
7	64.9 ± 24.2	65.0 ± 24.1	6 ± 8	73.8 ± 24.4	73.2 ± 25.1	8 ± 13

Table 5.2: Metrics obtained for the different datasets, along with the standard deviations over the 10 executions with different weight initializations (Bal. Acc.: balanced accuracy).

The datasets generated for this study are very simple and have no noise, still the model starts having difficulties with a sequential time dependency of $\mathtt{t} - 3$ (dataset 5) as shown in Table 5.2. We run those experiments for a training of three times more epochs to confirm this observation. With datasets 3 to 7, we evaluate the maximum depth of recurrence the model can capture in the data. We expected the model to have difficulties capturing longer time dependencies, and this is confirmed. It gives this model very little utility.

In addition, by just observing the rule grammar (Equation 5.5), the nested

architecture implies that a pattern with a time dependency of $\mathtt{t} - i$ needs to pass through the i previous layers by taking at least i neurons in the hidden nodes. This is highlighted in the exploded view of the model (see Appendix F).

The common strategy to counter the RNN limitations, is to use a Long-Short Term Memory (LSTM) model which has a hidden state that allows gradient to pass through the different steps [Hochreiter and Schmidhuber, 1997]. However, we were not able to find a solution to keep the interpretability of the model and equivalence with the rules with such LSTM cell due to its computation involving an hyperbolic tangent activation (breaking the binarization).

To conclude, combining two architectures like RNN and BNN implies combining their limitations. In this case, the training problems were emphasized even on the simplest datasets. The presented recurrent architecture comes with training limitations but also with rule grammar limitations due to the nested temporal expressions. While learning rules with a neural architecture from sequential data is very promising, we believe that neither this recurrent approach nor its potential extensions (LSTM for example) are the solutions to expressive sequential rules.

5.3 Convolutional Rule Neural Network (CR2N)

This section is a slightly modified version of our published paper [Collery et al., 2023].

In this section, we propose a novel differentiable fully interpretable method to discover both local and global patterns (i.e. catching a relative or absolute temporal dependency) for rule-based binary classification. It consists of a convolutional binary neural network with an interpretable neural filter and a training strategy based on dynamically-enforced sparsity. We demonstrate the validity and usefulness of the approach on synthetic datasets and on an open-source peptides dataset. Key to this end-to-end differentiable method is that the expressive patterns used in the rules are learned alongside the rules themselves.

5.3.1 Introduction

In this section, we bridge three domains (rule learning, sequential pattern mining and BNN) and introduce a binary neural network to learn classification rules on sequential data. We propose a differentiable rule-based clas-

sification model for sequential data where the conditions are composed of sequence-dependent patterns that are discovered *alongside* the classification rule. More precisely, we aim at learning a rule of the following structure: *if* pattern *then class* $= 1$ *else class* $= 0$. In particular we consider two types of patterns: local and global patterns [Aggarwal, 2002] that are in practice studied independently with a local and a global model. A local pattern describes a subsequence at a specific position in the sequence while a global pattern is invariant to the location in the sequence (Figure 5.3).

Examples of local and global pattern in the context of fraud detection are provided in Equation 5.7.

global	there are two consecutive transactions in the sequence where the destination account is labeled as suspicious.
local	the destination account is labeled as suspicious for the latest two transactions in sequence

(5.7)

The network, that we refer to as Convolutional Rule Neural Network (CR2N), builds on top of the base model RN presented in Chapter 4.

The contributions are the following: i) We propose a convolutional binary neural network that learns classification rules together with the sequence-dependent patterns in use. ii) We present a training strategy to train a binarized neural network while dynamically enforcing sparsity. iii) We show on synthetic and real world datasets the usefulness of our architecture with the importance of the rule grammar and the validity of our training process with the importance of sparsity. The code and datasets are publicly available at `https://github.com/IBM/cr2n`.

5.3.2 Architecture

5.3.2.1 Base Rule Model or Filter

The base rule model, invoked is composed of the three following consecutive layers: a StackedOR layer (Section 4.5.2), an AND layer and an OR layer (Section 4.2.1) as shown in (Figure 5.2).

The formal grammar that this architecture can express is specified with

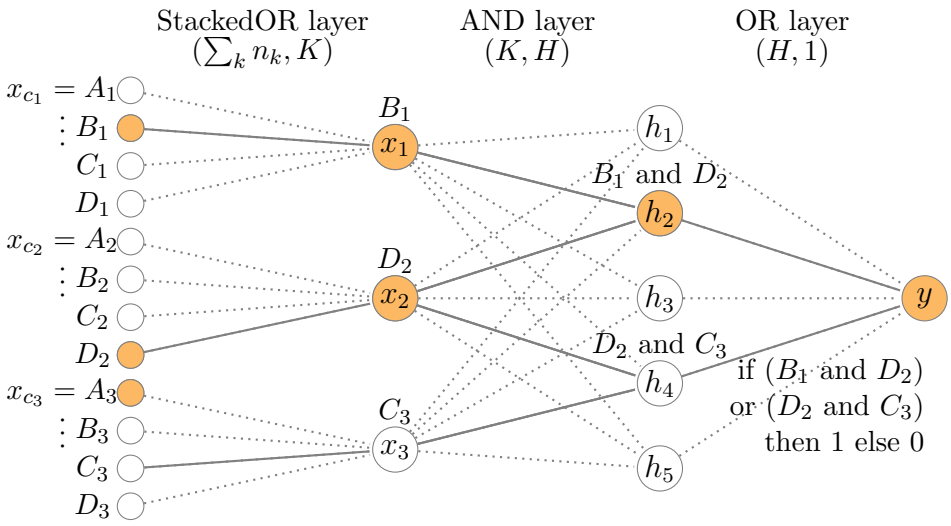


Figure 5.2: Example of a trained *base rule model* architecture for the rule *if (B_1 and D_2) or (D_2 and C_3) then 1 else 0* on 3 categorical variables x_{c_1}, x_{c_2} and x_{c_3} ($x_{c_k} \in \{A_k, B_k, C_k, D_k\}$). For simplicity, the truth value of $x_{c_1} = B_1$ is replaced by B_1 for example. Plain (dotted) lines represent activated (masked) weights. An example evaluation of the model is represented with the filled neurons (neuron=1) for the binary input $x_{c_1} = B_1$, $x_{c_2} = D_2$ and $x_{c_3} = A_3$.

the following production rules:

$$
\begin{array}{rcl}
\text{base expression} & \rightarrow & \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\
\text{conjunction} & \rightarrow & \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \\
\text{predicate} & \rightarrow & \text{categorical expression} \mid \text{literal} \\
\text{categorical expression} & \rightarrow & \text{categorical literal} \mid \text{categorical literal} \vee \text{categorical expression} \\
\text{categorical literal} & \rightarrow & \underline{x_c = \alpha_0^c} \mid \ldots \mid \underline{x_c = \alpha_{n_c}^c} \text{ (or simply } \underline{\alpha_0^c} \mid \ldots \mid \underline{\alpha_{n_c}^c}) \\
\text{literal} & \rightarrow & \underline{x_1} \mid \underline{x_2} \mid \ldots
\end{array}
\tag{5.8}
$$

As in Chapter 4, this grammar is also limited by the model architecture: *conjunction* contains at most one occurrence of each *predicate* and the total number of *conjunction*(s) is bounded by the number of hidden nodes.

5.3.2.2 Convolutional Rule Neural Network

The main contribution of this section is the extension of the base model RN to sequential data with a convolutional approach. We apply the base rule model as a 1D-convolutional window of fixed length $L \in \mathbb{N}$ over a sequence and retrieve all outputs as input for an additional disjunctive layer which we

refer to as the ConvOR layer as shown in Figure 5.3. The base rule model learns a disjunction of conjunctions over the window size length and the ConvOR layer indicates where along the sequence that logical expression is true. If the logical expression is to be applied all along the sequence then it can be described as a *global pattern*, otherwise the learned pattern represents a *local pattern*.

The model input is now of size $\sum_k L \times n_k$ and output of StackedOR layer (or input of the AND layer) is $L \times K$. Other dimensions are not impacted. For simplicity in the following, K is fixed to 1 i.e. input data is composed of one categorical variable evolving sequentially (a simple symbolic sequence). The method is still valid for complex symbolic sequence with $K > 1$ with possible limitations on the expressivity as mentioned in Section 4.5.2. Figure 5.2 is also still valid with a change of index, k is now referring to the position in the window of size L.

With this approach, different sequence-dependent expressions can be extracted and their nature depends on the weights of the ConvOR layer (Figure 5.3). If all the weights, $\boldsymbol{W}_{conv}$, of the ConvOR layer are activated (i.e. equal to 1), the logical expression learned by the base model is valid in all the sequence (true at least in one window): a *global* pattern is learned. If only some of the weights of the ConvOR layer are activated, the logical expression learned by the base model is valid only in the window associated to that weight: a *local* pattern is learned. The base model logical expression is modified accordingly to match that shift (see example in Figure 5.3 with a shift of 3 sequential steps).

The obtained weights thus translate to a rule grammar with the following production rules:

$$\begin{aligned} \text{rule} &\rightarrow \textit{if}\ \text{expression}\ \textit{then}\ \textit{class} = 1\ \textit{else}\ \textit{class} = 0 \\ \text{expression} &\rightarrow \text{local pattern} \mid \text{global pattern} \end{aligned} \tag{5.9}$$

`t` is defined as the last observation in a sequence, in the same manner as in Section 5.2. A, B, C and D are toy binary input possible values for our categorical variable x_c (they cannot be activated simultaneously at the same position `t` in the sequence). With those definitions, we list below examples of different sequence-dependent expressions that can be expressed with the proposed architecture (see Figure 5.3):

A **local pattern** is an expression composed of predicates that are true at a specific position i, for example A *at* `t`*-15*. Based on Equation 5.8 we have:

$$\begin{aligned} \text{local pattern} &\rightarrow \text{base expression} \\ \text{predicate} &\rightarrow \text{categorical expression}\ \textit{at}\ \texttt{t} - \underline{i} \mid \text{literal}\ \textit{at}\ \texttt{t} - \underline{i}. \end{aligned} \tag{5.10}$$

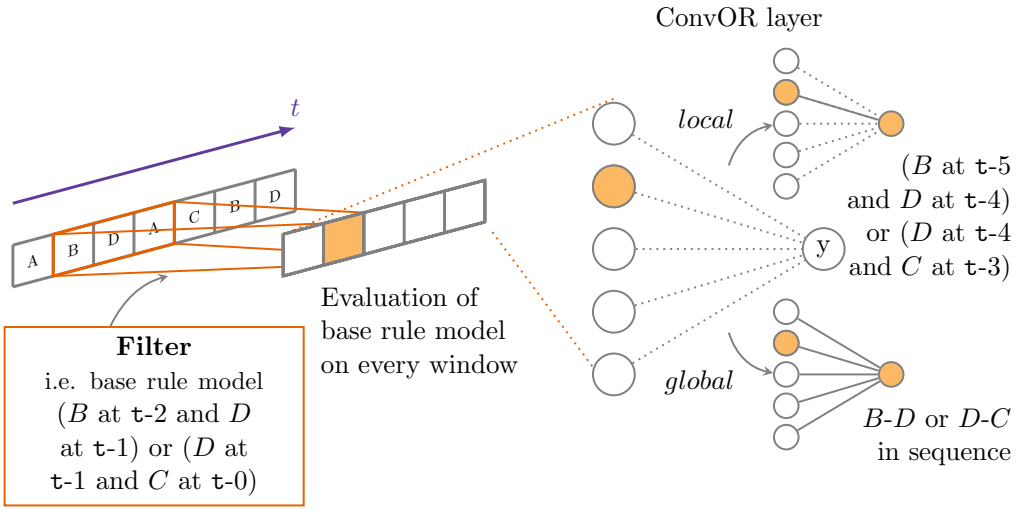


Figure 5.3: Example of a trained CR2N architecture. The base rule model or filter is applied as 1D-convolutional window over the sequence (i.e. sliding window). The resulting boolean values are given as input of the ConvOR layer which indicates through its activated weights where along the sequence the expression learned by the base model is true. The output of the ConvOR layer is mapped to the label of the sequence y. For local patterns, the base model expression needs to be shifted accordingly to the ConvOR layer weights. For a real-domain application like fraud detection, by providing meaning to B, C and D, we could have for example if "receiving a transaction of amount X"(B) is followed by "emitting a transaction of amount X" (D) or "emitting a transaction of amount X"(D) is followed by "closing the bank account"(C) then class=fraud.

A **global pattern** is an expression describing the presence of a pattern anywhere in the sequence, for example *B-D in sequence* is a global pattern where "−" sign refers to "followed by" and "∗" correspond to any unique literal (equivalent to $\forall i \in [0; T-2]$, *B at* `t`*-i-1 and D at* `t`*-i*) . If inputs are sequences of characters, global patterns can be compared to simple regular expressions supporting the logical OR (metacharacter'[ ]'). Based on Equation 5.8 we have:

$$\begin{aligned} \text{global pattern} &\rightarrow \text{base expression} \\ \text{conjunction} &\rightarrow \text{predicate} \mid * \mid \text{predicate} - \text{conjunction} \end{aligned} \tag{5.11}$$

Additional special cases can be pointed out such as the learning of a global pattern over an interval (e.g. *B-*-D in window [*`t`*-6; *`t`*-3]*) or the learning of

sequence characteristics dependant expression such as $4 \leq len(sequence) \leq 6$ based on the sequence length (not shown on Figure 5.3 but it corresponds to a specific case where the base model has learned an always true rule). Also, it is important to note that *base expression* and *conjunction* lengths in both grammars are bounded by the fixed window size L. The full grammar of the rule language that the model can represent is available in Appendix C.4.

To ensure full equivalence between the model and rule, sequences boundaries need to be considered, especially for global patterns. All sequences are padded on both ends with a sequence of 0 of size $L-1$ (not shown for simplicity on Figure 5.3). Also sequences of different lengths are supported by creating a model based on the maximal available sequence length M in the data and padding shorter sequences with a sequence of 0 of appropriate length. ConvOR layer input size is then $M+L-1$.

With this one architecture, we can model both local and global patterns. However for optimization reasons detailed below, we choose to differentiate the two into two distinct models: a **local** and a **global model**. The ConvOR layer weights for the global model are set and fixed to 1 during training.

5.3.3 Training

In addition the training strategy from Chapter 4 that is kept unchanged (Section 4.2.2), we propose to improve the training by enforcing sparsity dynamically. The model parameters Θ of CR2N are $\boldsymbol{W}^1_{stack}, \ldots, \boldsymbol{W}^K_{stack}, \boldsymbol{W}_{and}, \boldsymbol{W}_{or}$ and $\boldsymbol{W}_{conv}$.

Loss function Loss function from Equation 4.11 is still valid and in practice, we use $\lambda = 10^{-5}$. The regularization term Π , or *penalty*, that evaluates the number of terminal conditions in the rule represented by CR2N depends on the penalty of the base rule model. First the base rule model penalty Π_{base} is defined before expressing Π.

$$\boldsymbol{\Pi}_{stack} = \begin{bmatrix} \boldsymbol{W}^1_{stack}\mathbf{1} \\ \vdots \\ \boldsymbol{W}^K_{stack}\mathbf{1} \end{bmatrix}, \qquad \Pi_{base} = \boldsymbol{W}_{or}\boldsymbol{W}_{and}\boldsymbol{\Pi}_{stack} \tag{5.12}$$

For optimizing Π for local patterns, we have to minimize the activated ConvOR layer weights. For global patterns, we want them to all be activated. A condition could be set on the sum of ConvOR layer weights (Equation 5.13) to shift from one optimization problem to the other but with loss of continuity and thus differentiability (interesting values of τ being $M+L-1$, the ConvOR layer input size, that would correspond to all ConvOR layer weights being

equal to 1, or $M - L + 1$ that would allow for $2(L-1)$ weights to be 0, and corresponds to the padding required for properly accounting for sequence boundaries (Section 5.3.2.2)).

$$\Pi_{local} = \Pi_{base}\boldsymbol{W}_{conv}\mathbf{1}, \qquad \Pi_{global} = \Pi_{base}, \qquad \Pi(\Theta)^* = \begin{cases} \Pi_{global} & \text{if} \boldsymbol{W}_{conv}\mathbf{1} \geq \tau \\ \Pi_{local} & \text{otherwise} \end{cases} \tag{5.13}$$

Due to non continuity of Π^* in Equation 5.13, we choose to have two models with the same architecture for the two cases: the *local* and the *global model* respectively more relevant for their associated pattern. For the local model, all weights are trainable and $\Pi(\Theta) = \Pi_{local}$. For the global model, weights in the ConvOR layer are fixed and set to 1, and $\Pi(\Theta) = \Pi_{global}$.

Enforced sparsity Sparsity of the model is crucial to learn concise expressions, the model needs to generalize without observing *all* possible instances at training time. The first requirement for that matter is sparsity in the base rule model. In addition to the regularization term in the loss function, we propose to use a sparsify-during-training method [Hoefler et al., 2021] and dynamically enforce sparsity in weights from 0% to an end rate r_f set to 99% in our case [Lin et al., 2020]. Sparsify-during-training method can also benefit the quality of the training in terms of convergence by correcting for approximation errors due to premature pruning in early iterations but is highly dependant on the sparsification schedule [Hoefler et al., 2021].

As illustrated in Algorithm 4, every 16 iterations s and for a total of s_f training iterations, every trainable weight in Θ is pruned with a binary mask, $\boldsymbol{B}$ (of size of its associated weight and applied with Hadamard product ($\odot$)) [Lin et al., 2020, Zhu and Gupta, 2017]. We propose a mask based on the maximum of weight magnitude $\boldsymbol{C}$ and pruning rate r [Zhu and Gupta, 2017] making the assumption that it contributes to generalization (Equation 5.14). This strategy can be more aggressive than state-of-art contributions [Lin et al., 2020] due to its dependency to the $\boldsymbol{C}$ maximum value. During training, the model with the highest prediction accuracy on validation dataset and the highest sparsity (evaluated at each epoch) is kept.

$$r = r_f - r_f\left(1 - \frac{s}{s_f}\right)^3, \qquad B_{i,j} = |C_{i,j}| \geq r \times \max(\boldsymbol{C}), \qquad \hat{\boldsymbol{W}} = \boldsymbol{W} \odot \boldsymbol{B} \tag{5.14}$$

Additional training optimizations have been tested out such as for example replacing the latent weights with an optimizer for BNN [Helwegen et al.,

Algorithm 4: Training procedure with enforced sparsity

Data: Training dataset $\mathcal{D}$, Learning rate: η, Batch size: B, Number of epochs: E, Pruning period: p, Pruning end rate: r_f
Result: Trained neural network model
Initialize neural network parameters;
Initialize binary mask $\boldsymbol{B}$ to 1;
$s_f = E \times B$; ▷ `number of iterations`
for $i \leftarrow 1$ **to** E **do**
  for $j \leftarrow 1$ **to** $\frac{N}{B}$ **do**
    $s = i \times j$; ▷ `training iteration`
    if p *divides* s **then** ▷ `trigger mask update`
      Compute masks $\boldsymbol{B}$ for every model weights parameters and for end rate r_f (Eq. 5.14);
    Apply masks $\boldsymbol{B}$ on model weights parameters: $\hat{\boldsymbol{W}} = \boldsymbol{W} \odot \boldsymbol{B}$;
    Randomly sample a mini-batch $\mathcal{D}_{\text{batch}}$ of size B from $\mathcal{D}$;
    Forward pass: Compute predictions on mini-batch using the pruned model parameters $\hat{\boldsymbol{W}}$;
    Compute the loss between predictions and actual targets (Eq. 4.11);
    Backpropagation: Compute gradients of the loss with respect to the parameters;
    Update the parameters using Adam optimizer with learning rate η;
  Evaluate model on validation dataset;
  Keep model with best loss on validation dataset;
return *Best model*;

2019, Geiger and Team, 2020] introduced in Section 4.2.2, adding a scheduled cooling on the sigmoid of the binarized weights, alternating the training of each layer every few epochs [Qiao et al., 2021] or using a learning rate scheduler. Those techniques are not detailed in more depth here but could be of interest for improving results on specific datasets.

Training on sequences of variable lengths There are different ways to handle sequences and batches of sequences of variable lengths. The first one is to create batches of same length sequences. This strategy assumes at each weights update that the searched patterns are independent from the sequences length. The training is likely to be distorted for that reason,

but also because of the smaller batches resulting from the split by length that will generate noisy gradient estimates. However this is a strategy that is commonly used for example in the field of Natural Language Processing (NLP). The second is to pad sequences to be of the same length and mask the appropriate neurons once the sequential data processing is complete. In our case this would correspond to masking parts of the input of the conv layer. In practice in the following the first option is implemented with batches of same length sequences.

5.3.4 Experiments

In order to evaluate the validity and usefulness of this method, we apply it to both synthetic datasets and UCI membranolytic anticancer peptides dataset [Grisoni et al., 2019, Dua and Graff, 2017].

#	**Ground Truth**	**+class**	π
1	C at `t`-4	14.2	1
2	A at `t`-6 and C at `t`-4	1.5	2
3	(A at `t`-6 and C at `t`-4) or (B at `t`-5 and C at `t`-3)	3.6	4
4	B-D in sequence	20.4	2

Table 5.3: Ground truths applied on sequences of letters (A to F) to generate synthetic unbalanced datasets 1, 2, 3 and 4 along with the proportion of the positive class (in %) and π, the number of conditions in the expression. `t` refers to the position when the last observation in a sequence was made. Balanced datasets with same ground truths are generated and referred to as the dataset number followed by the letter b (Appendix A).

Synthetic Datasets We propose 8 synthetic datasets based on 4 ground truth expressions with both balanced and unbalanced class distribution for discovering simple binary classification rules with local or global patterns as shown in Table 5.3. There are 1000 sequences of letters (A to F) of different lengths from 4 to 14 letters in each of them (Mean around 9±3). Generation is detailed in Appendix A.

Peptides Dataset Besides the synthetic datasets, real-world UCI anticancer peptides dataset composed of labeled one-letter amino acid sequences, is used [Grisoni et al., 2019, Dua and Graff, 2017]. The multi-classification

Model	Window	Pruning	Accuracy	Bal. Acc.	Penalty (Π)	Epoch
global	3	No	88.9 ± 5.0	75.3 ± 12.7	58.7 ± 35.9	105 ± 60
		Yes	85.3 ± 5.3	67.3 ± 14.3	19.4 ± 15.9	23 ± 20
		30	88.7 ± 5.0	75.4 ± 12.9	46.9 ± 28.0	48 ± 22
	6	No	**91.2** ± 1.0	**81.8** ± 2.0	220.0 ± 56.5	92 ± 82
		Yes	89.5 ± 3.6	77.6 ± 9.3	132.7 ± 93.9	33 ± 15
		30	**91.0** ± 1.3	**82.0** ± 2.8	**97.3 ± 60.6**	71 ± 19
local	3	No	90.0 ± 3.7	78.7 ± 9.7	694.3 ± 269.3	77 ± 48
		Yes	**90.3** ± 1.5	**81.6** ± 1.6	**885.2** ± 328.7	25 ± 8
		30	89.2 ± 5.2	76.3 ± 13.2	674.7 ± 483.9	49 ± 13
	6	No	**92.1** ± 1.0	**83.5** ± 2.5	**3.9k ± 1.4k**	91 ± 58
		Yes	88.0 ± 4.7	74.6 ± 12.4	1.0k ± 1.0k	34 ± 16
		30	**91.7** ± 1.5	**83.5** ± 2.6	1.8k ± 0.6k	49 ± 10

Table 5.4: Performance metrics obtained for the different models, window size and pruning strategy on the peptides dataset, along with the standard deviations over the 10 executions with different weights initializations. (Bal. Acc.: balanced accuracy, Epoch: best epoch).

problem is transformed into a binary classification problem is the same manner as [Nwegbu et al., 2022] (see Appendix D). Sequence lengths are from 5 to 38 letters (Mean: 17 ± 5.5) and positive class distribution is 79%.

Experimental Setting Experimental setting from previous section is still valid (Section 5.2.4), with the following changes. The hidden size in the base rule model is set to the double of the input size of the AND layer (which is the window size of the convolution). At each epoch (200 in total), we evaluate the model against the validation dataset and keep the model with the highest accuracy and in case of equality the model with lowest penalty. We run the experiments with two different window sizes (3 and 6) for the CR2N convolution filter size. We compare the two versions of the architecture: the local and global models described in Section 5.3.3 and study three different dynamic pruning strategies: none, dynamic enforced sparsity from epoch=0 and from epoch=30 (arbitrary).

5.3.5 Results and Discussion

Rule grammar and expressivity The importance of the rule model expressivity can be seen concretely by comparing the different patterns the local and global models have learned for dataset 3b for example:

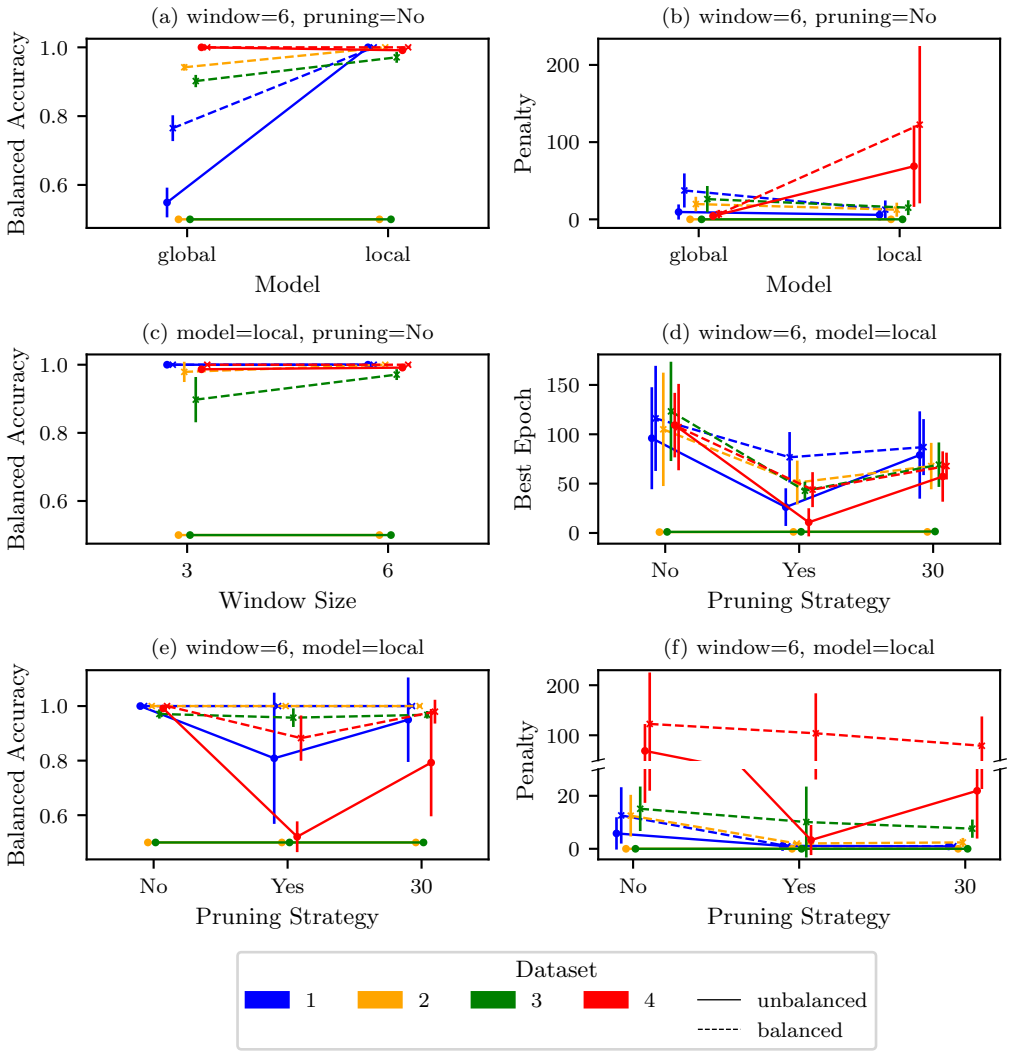


Figure 5.4: Representations of key results obtained on the synthethic datasets. Error bars represent the standard deviations over the 10 executions with different weights initializations.

1. (A or B)-*-(C)-*-*-(A or B or C or D or E or F) in sequence (global, no pruning, window size=6), and
2. (B at t-5 and C at t-3) or (A at t-6 and C at t-4) (local, pruning, window size=6).

In the first case, the grammar is not appropriate to model the data (as a reminder, the global model is a constrained version of the local model) as opposed to the local model that learned the perfect rule. In practice, on real

data, obtained patterns, such as "if (D or E or G or H or I or N or Q or T or Y)-*-*-*-(D or E or G or I or N or Q or S or T or V or Y) in sequence" obtained for labeling 'inactive-virtual' peptides, can be explored further by a domain expert. Black box approaches do not provide such insights.

The importance of the rule model expressivity is also highlighted by comparing experiments with local or global models and experiments with different window sizes. First of all, the accuracy of the local model is higher compared to the global model on balanced synthetic datasets 1, 2 and 3 (Figure 5.4(a)). For balanced and unbalanced dataset 4, both models achieve very high accuracies ($> 95\%$). However, as shown on Figure 5.4(b), it is at the cost of rule complexity for the local approach with averaged penalty values higher than 60 (and standard deviation higher than 50) compared to lower than 10 for the global model (and standard deviation lower than 5). It points out that the local model in that case requires on average at least more than 6 times more terminal conditions in the learned rule than the global model for comparable accuracies, but also that the weights initial states have a huge impact on the rule complexity when the rule grammar is not expressive enough (with no pruning). Those results are confirmed on real-world dataset with the peptides dataset, accuracies between the local and global models especially for a window size of 6 are comparable. However there is an order of magnitude difference for the penalty, global approach being more concise. It is important to note that by design the global approach has less weights to train and thus a much lower maximum penalty.

Datasets 2b and 3b benefit from a bigger window size (expected for dataset 3/3b due to ground truth pattern size) as shown in Figure 5.4(c). Accuracies are also higher with window size 6 than 3 for the peptides dataset at the cost of also higher penalties (Table 5.4).

The more expressive the model is, i.e. the more patterns it can model, training limitations aside, the better the performance. Of course common black box models with no such 1-1 rule mapping constrained architecture would reach 100% accuracy, but it is that mapping in particular that makes the model relevant, expressive and fully interpretable. Also, the best performances in accuracy for the peptides dataset ($\sim 91\%$) are comparable to the best results ($\sim 92\%$) obtained from classification with single kernels when applied to that same dataset in [Nwegbu et al., 2022], our model providing an additional fully-interpretable property. The presented model is also flexible due to its logical equivalence and can be inputted into other logical layers for deeper architectures to extend the rule grammar [Beck and Fürnkranz, 2021a]. It can also be extended for time series with well-chosen filter size for instance. Other rule grammar extensions can be inspired by Linear Temporal Logic domain and regular expression pattern mining [De Giacomo et al.,

2022]. However the more expressive the model is the more effort is required for training and rule complexity.

Sparsity and training strategy The importance of model sparsity is pointed out by the experiments with different pruning strategies. First, looking at training scenarios, both on synthetic and peptide datasets, experiments with sparsity-during-training approaches reach the best model faster on average than without (lower best epoch Figure 5.4(d)). Then, regarding the performance in terms of accuracy, we can differentiate two cases: balanced and unbalanced datasets. Training of unbalanced datasets is more affected by the aggressive dynamic pruning strategy than balanced datasets. For example, we observe a drop on average of around 0.2 in accuracy for dataset 1 compared to balanced dataset 1 (Figure 5.4(e)). The pruning strategy starting after 30 epochs is preferred in both cases. Average accuracies with a pruning strategy not starting immediately (30 epochs) are comparable to the ones obtained without pruning for balanced datasets. In terms of rule complexity, penalty values are lower with pruning and even lower when starting after 30 epochs in most cases (Figure 5.4(f)).

With our pruning strategy (Equation 5.14), we make the assumption that lower positive *loc* values are associated to overfitting or redundancy by taking into account that values closer to 0, i.e. on the sigmoid slope, are more likely to shift thus less 'certain'. As pointed out in early work by [Prechelt, 1997], the dynamic pruning strategy helps to overcome possible lower generalization ability compared to a fixed pruning which could explain cases of better performance (peptide dataset local model window size of 3 for example). A different pruning strategy based on a generalization loss was proposed to characterize the amount of overfitting [Prechelt, 1997]. While this strategy is relevant in more general cases and can be applied to many different networks, our strategy is tailored for minimizing positive trainable parameter values.

Sparsity of the model is also induced via the regularization term Π in the loss function $\mathcal{L}$ (Equation 4.11). With this method, the importance of sparsity is controlled in with a regularization coefficient (λ) but the final sparsity is not directly configurable. A dynamic pruning strategy on the other hand, is easier to control for both target sparsity and accuracy but is highly dependent on the pruning schedule [Hoefler et al., 2021].

An interesting point is made by [Hoefler et al., 2021] about the convolutional operator that 'can be seen as a sparse version of fully-connected layers'. That level of forced sparsity in our model is therefore defined by the fixed window size model parameter with respect to the maximum sequence

length. The ideal sparser window size would be the size of the maximal temporal hidden pattern in data that can only be approximated with external or expert knowledge and/or tuned with trial and error.

With or without a dynamic pruning strategy, for highly unbalanced datasets (2 and 3), experiments have shown that the training strategy of the model is not suitable. Indeed most of them, label everything with the majority class (50% balanced accuracy). It corresponds to the specific case of learning an empty rule (penalty=0) (Figure 5.4(a,c,e)). For unbalanced datasets 1 and 4, their best models do not reach on average the same accuracies as in their balanced versions.

Overall this training strategy is both the key and the main limitation of our approach: it can provide a sharp concise rule with minimal redundancy and simplified logical expression but it is highly dependent on numerous model, training and pruning parameters and is not suited as is for highly unbalanced datasets.

To conclude, we presented a 1D-convolutional neural architecture to discover local and global patterns in sequential data while learning binary classification rules. This architecture is fully differentiable, interpretable and requires sparsity that is enforced dynamically. One main limitation is its dependence to the window size and sparsity scheduler parameters. Further work will consist in integrating this block into more complex architectures to augment the expressivity of the learned rules as well as extending it for multi-classification.

5.4 Conclusion

In conclusion, this chapter explored and evaluated two approaches for neural-based rule learning for sequences. The first approach, based on a recurrent architecture, exhibited limitations inherent to traditional recurrent neural networks (RNNs). Despite its potential, this recurrent approach lacked significant room for improvements within a fully interpretable context (inspiration from LSTM for example). Consequently, an alternative approach was pursued. The second approach centered around a convolutional architecture, demonstrated the ability to effectively learn two distinct types of temporal patterns. Additionally, a novel training strategy incorporating dynamically enforced sparsity was introduced to obtain concise and accurate rules. Also with well-chosen filter size, the architecture presented for sequential data could be adapted to time series. This model is of great interest as it could be extended to further increase the expressiveness of the learned rules.

Chapter 6

Expressive neural-based rule learning

Contents

6.1 Introduction

In the last two chapters, we defined and proposed neural architectures for rule learning on binary features with RN and on sequential data with CR2N for example. In this chapter, we build on top of them to extend the grammar of the learned rules. By deepening the architecture and plugging in rule learning neural blocks into one another, we get closer to the grammar available in BRMS.

In the first section, we look into the learning of rules with predefined aggregation functions. Learning functions with neural networks has been a research interest for some time now [Udrescu and Tegmark, 2020]. From basic linear combinations to complex mathematical experessions [Udrescu and Tegmark, 2020] neural networks are suitable (or not) for these applications [Thompson et al., 2022, Willard et al., 2022]. However what we are interested in is not learning the mathematical function itself but in learning a model that selects appropriate aggregation functions from a catalog (predefined list), as a user would in a BRMS, and learns its arguments. We can mention for instance *min*, *max*, *sum* and *product* as common aggregation functions that can be applied on sequential data or sets for example. In practice, these functions are used extensively by BRMS business users because they provide key quantitative information to model their business policy. In the context of fraud detection, learning the following condition "minimum amount of the 3 last transactions is higher than 1000 euros" could be of interest for example, for classifying fraudulent transactions.

Then in a second part of the chapter, we deepen the architecture and present a model that can express rules based on multiple filters on sequential data which implies that a rule can be based on multiple local and/or global patterns as defined in Chapter 5. Changes in the dimensions of the model lead to learning difficulties. Two new implementations of the OR and AND layers that approximate the max operation in different ways are proposed and then compared. This architecture is a next step towards more expressive rules. To the best of our knowledge, there is no existing models able to learn such expressive classification rules in the literature.

As in Chapter 5, we consider a supervised binary classification problem where we aim to predict a binary output label y, based on multivariate sequential input data $\boldsymbol{X}$. We are provided with a training dataset $\mathcal{D} = \{(\boldsymbol{X}^{(1)}, y^{(1)}), (\boldsymbol{X}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{X}^{(N)}, y^{(N)})\}$, where N is the number of training samples and $\mathcal{D}$ consists of N pairs of multivariate sequential data $\boldsymbol{X}^{(i)}$ and their corresponding binary labels $y^{(i)}$. A multivariate sequence $\boldsymbol{X}$ (i.e. $\boldsymbol{X}^{(i)}$) of D features and of length T is defined as $\boldsymbol{X} = (\boldsymbol{x}_0, ..., \boldsymbol{x}_{T-1})^T \in [0,1]^{T \times D}$, where $\boldsymbol{x}_t$, $\forall t \in \{0, ..., T-1\}$, is the t^{th} observation of all D variables. A

binary output label y (i.e. $y^{(i)}$) is represented as $y \in \{0, 1\}$.

We also keep the previously introduced $\mathtt{t}$ as the position when the last observation in a sequence was made. With $\mathtt{t}$ being our reference, $\mathtt{t} - i$ refers to the moment of the i^{th} observation before $\mathtt{t}$ with $i \in \{0, ..., T-1\}$).

The models presented can also be extended to a set of N sequences of $\boldsymbol{x}$ of a fixed number of feature D and of different lengths T_n.

6.2 Simple Convolutional Rule Neural Network with Aggregates (s-CR2NA)

We extend the CR2N model to introduce the learning of rules with predefined aggregation functions on sequences.

6.2.1 Architecture

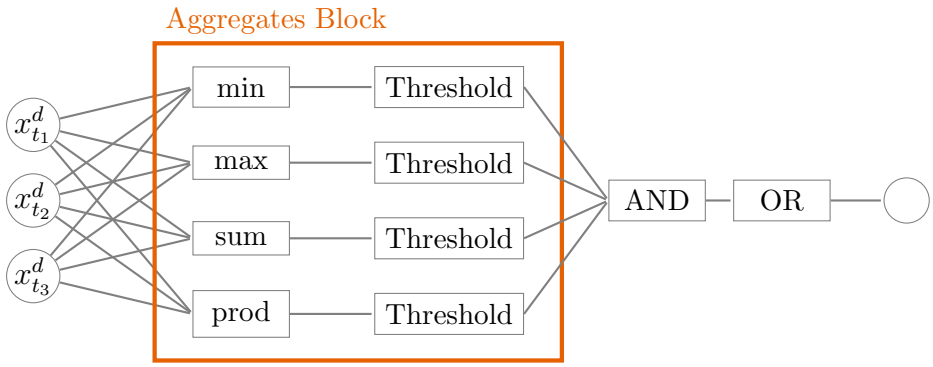


Figure 6.1: Example of filter proposed to learn rules with predefined aggregation functions on input $\boldsymbol{x}^d$ when used with the model presented in Figure 6.2. This example specifies min, max, sum and product as predefined aggregation functions.

Base Rule Model or Filter The AND and OR layers of the base rule model (a.k.a. the filter) are kept untouched in this extended architecture. As input of the AND layer, a new block, referred to as the aggregates block, applies J predefined aggregation functions to the input data before binarizing the resulting values with a Threshold layer as introduced in Section 4.5.1.

$\bar{\boldsymbol{x}}^d_S$ or $(\boldsymbol{x}^d_t)_{t \in S}$ refers to the values of feature d in the subsequence of $\boldsymbol{X}$ defined by the set S of contiguous indices (corresponds to the window

on which the filter is applied). The aggregates block on a window S with $\{f_1, \ldots, f_J\}$ the set of predefined aggregation functions, is implemented as follows.

$$\boldsymbol{a}_S = \sigma\left(\frac{1}{\theta}\left(\begin{bmatrix} f_1(\bar{\boldsymbol{x}}_S^1) \\ \ldots \\ f_J(\bar{\boldsymbol{x}}_S^D) \end{bmatrix} - \boldsymbol{b}\right)\right) \tag{6.1}$$

σ is the sigmoid activation function applied component-wise with temperature θ and learnable bias $\boldsymbol{b}$. In practice, we use $\theta = 0.1$.

A Heaviside step function replaces the sigmoid in the same manner as in Section 4.5.1 for validation, testing and rule extraction. As shown in Section 4.5.1, hidden neuron $\boldsymbol{a}_{S,1}$ represent the logical expression $f_1(\bar{\boldsymbol{x}}_S^1) > \boldsymbol{b}_1$.

In the following we will consider four basic aggregation functions: *min*, *max*, *sum* and *product*. However, the method can be applied to any function that maps a set of data to a (numerical) value. Resulting values are combined and provided as input to the base rule layers as shown in Figure 6.1

The new filter (Figure 6.1) associated grammar production rules are defined in Equation 6.2.

$$\begin{aligned} \text{base expression} &\rightarrow \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\ \text{conjunction} &\rightarrow \text{aggregate} \mid \text{aggregate} \wedge \text{conjunction} \\ \text{aggregate} &\rightarrow \text{aggfunction } \textit{of } \underline{x^d} \textit{ is greater than } \underline{\text{value}} \\ \text{aggfunction} &\rightarrow \textit{min} \mid \textit{max} \mid \textit{sum} \mid \textit{product} \end{aligned} \tag{6.2}$$

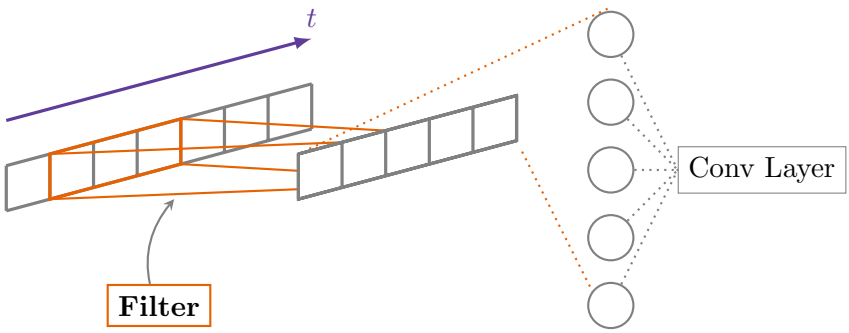


Figure 6.2: Base architecture of s-CR2NA model with a placeholder for the conv layer and filter.

Conv layers In addition to the ConvOR layer introduced in the previous chapter, we define the ConvAND layer. In the same manner as the ConvOR, the ConvAND layer indicates where along the sequence the logical expression learned in the filter must be true but combines those positions with a conjunction. Together with their local and global variants, there are 4 different *conv layers* to gather the filter outputs along the sequence. The base architecture of this model that we refer to as s-CR2NA is shown in Figure 6.2 and can take one conv layer as output layer. These conv layers, combined with the previously described filter architecture, augment the expressivity of the filter expression. With $\hat{S}$ the set of sets S_i of X of size L that corresponds to the selected windows by the conv layer, depending on the conv layer we can express the following expressions (Equation 6.3). Full grammar of the model rule language is available in Appendix C.5.

$$\begin{aligned} \text{localOR} &\rightarrow \text{base expression } \textit{over any of the windows: } \hat{S} \\ \text{localAND} &\rightarrow \text{base expression } \textit{over all of the windows: } \hat{S} \\ \text{globalOR} &\rightarrow \exists \underline{S} \textit{ of size } \underline{L} \textit{ in seq such that } \text{base expression} \\ \text{globalAND} &\rightarrow \forall \underline{S} \textit{ of size } \underline{L} \textit{ in seq, } \text{base expression} \end{aligned} \tag{6.3}$$

Table 6.1 highlights their role by comparing their impact on a simple filter expression $f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$ both with the mathematical expressions and textual equivalents.

Conv Layer	**Math**	**Text**
Local OR	$\bigvee_{S \in \hat{S}} f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$	f of d is greater than b over any of the windows S.
Local AND	$\bigwedge_{S \in \hat{S}} f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$	f of d is greater than b over all of the windows S.
Global OR	$\exists S$ of size L $\lvert f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$	There are L consecutive elements in sequence x, where f of d is greater than b.
Global AND	$\forall S$ of size L, $f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$	For all L consecutive elements in sequence x, f of d is greater than b.

Table 6.1: Possible expressions that can be learned with different conv layers for a filter that corresponds to the expression $f(\bar{\boldsymbol{x}}_S^d) > \boldsymbol{b}$.

6.2.2 Training

As opposed to the training strategy in use in previous chapter (Chapter 5), enforced sparsity is not applied in these experiments. We believe that adding a pruning strategy may help optimizing the complexity of the rules and help

and fasten the training. However, finding satisfactory configuration for the enforced sparsity presented in Section 5.3.3 is challenging and improvements were modest, it is therefore omitted. Also, the learning rate η of Adam optimizer is set to 0.01. The rest of the training strategy is kept unchanged and summarized in Algorithm 5 (Section 4.2.2). The model parameters Θ of s-CR2NA are $\boldsymbol{W}_{and}, \boldsymbol{W}_{or}, \boldsymbol{W}_{conv}$ and $\boldsymbol{b}$.

Algorithm 5: Training procedure

Data: Training dataset $\mathcal{D}$, Learning rate: η, Batch size: B, Number of epochs: E
Result: Trained neural network model
Initialize neural network parameters;
for $i \leftarrow 1$ **to** E **do**
  for $j \leftarrow 1$ **to** $\frac{N}{B}$ **do**
    Randomly sample a mini-batch $\mathcal{D}_{\text{batch}}$ of size B from $\mathcal{D}$;
    Forward pass: Compute predictions on mini-batch using the current model parameters;
    Compute the loss between predictions and actual targets (Eq. 4.11);
    Backpropagation: Compute gradients of the loss with respect to the parameters;
    Update the parameters using Adam optimizer with learning rate η;
  Evaluate model on validation dataset;
  Keep model with best loss on validation dataset;
return *Best model*;

Loss function The loss function from Equation 4.11 is still valid ($\lambda = 10^{-4}$). s-CR2NA being very close to the architecture of CR2N, the computation of their associated regularization terms Π is almost identical as shown in Equation 6.4. The number of terminal conditions of the base rule model Π_{base} now corresponds to the penalty of RN model.

$$\Pi_{base} = \boldsymbol{W}_{or}\boldsymbol{W}_{and}\mathbf{1}, \qquad \Pi_{local} = \Pi_{base}\boldsymbol{W}_{conv}\mathbf{1}, \qquad \Pi_{global} = \Pi_{base} \tag{6.4}$$

For the local model, all weights are trainable and $\Pi(\Theta) = \Pi_{local}$. For the global model, weights in the Conv Layers are fixed and set to 1, and $\Pi(\Theta) = \Pi_{global}$.

#	Type	Ground Truth	+class	π
1	local	$\max_s F_0 > 0.8$ *over window:* s_{-1}	49.2	1
2	localOR	$\max_s F_0 > 0.9$ *over any of the windows:* s_{-1}, s_{-4}	43.9	2
3	localAND	$\max_s F_0 > 0.9$ *over all of the windows:* s_{-1}, s_{-4}	62.7	2
4	globalOR	$\exists s$ *of size* 3 *in seq such that* $\sum_s F_5 > 2$	55.0	1
5	globalOR	$\exists s$ *of size* 3 *in seq such that* $\prod_s F_5 > 0.25$	52.8	1
6	globalAND	$\forall s$ *of size* 3 *in seq,* $\min_s F_0 > 0.05$	64.5	1

Table 6.2: Ground truths applied on numerical input features $F_0, \ldots, F_9 \in [0, 1]^{10}$ to generate the 6 synthetic datasets along with the proportion of the positive class (in %) and the penalty π of the expression.

Training on sequences of variable lengths Unlike experiments in Chapter 5.3, sequences are padded to be of the same length. Data resulting from applying the filter on the padded values, is then masked and replaced by a default value before being provided to the conv layer. The default value depends on the nature of the conv layer. For Local OR and Global OR conv layers default value is 0 (i.e False) and for Global AND conv layer default value is 1 (i.e. True). For Local AND layer the classical usage is to use 0 as a default value (used in practice). However, 1 can be chosen as default value with a consequence on the learned patterns as it will imply an implicit condition on the length of the sequence. Also batch normalization for Threshold layers takes into account the padding (ignores the padded values).

6.2.3 Experiments

In order to evaluate the validity and usefulness of this method, we apply the model on synthetic datasets.

Synthetic Datasets We propose 6 synthetic datasets for discovering simple binary classification rules with local or global patterns involving aggregation functions as shown in Table 6.2. There are 6000 sequences of 10 numerical features (F_i) ranging from 0 to 1. Sequences are of different lengths from 4 to 14 elements each. Generation is detailed in Appendix A. The model(s) with the conv layers corresponding to the dataset ground truth pattern is(are) manually chosen for the experiments.

#	Local OR	Local AND	Global OR	Global AND
1	99.7 ± 0.3	100.0 ± 0.0	-	-
2	100.0 ± 0.0	-	-	-
3	-	99.6 ± 0.2	-	-
4	-	-	98.5 ± 0.1	-
5	-	-	100.0 ± 0.0	-
6	-	-	-	100.0 ± 0.0

Table 6.3: Average accuracies obtained for the different models and datasets, along with the unbiased standard errors of the average over the 10 executions with different weights initializations.

#	Local OR	Local AND	Global OR	Global AND
1	1.9 ± 0.3	1.1 ± 0.1	-	-
2	3.2 ± 0.6	-	-	-
3	-	4.0 ± 0.6	-	-
4	-	-	8.8 ± 1.0	-
5	-	-	10.1 ± 0.9	-
6	-	-	-	2.1 ± 0.3

Table 6.4: Average penalties Π obtained for the different models and datasets, along with the unbiased standard errors of the average over the 10 executions with different weights initializations.

Experimental Setting All datasets are partitioned in a stratified fashion with 60% for training, 20% for validation and 20% for testing datasets and we use a batch size of 200 sequences. The hidden size in the base rule model is set to 20. Output sizes of the threshold layer and of the filter are 1. More details on experimental setting can be found in Appendix E. At each epoch (400 in total), we evaluate the model against the validation dataset and keep the model with the lowest loss. For each experiment, we run the algorithm 10 times with different weights initializations. Resulting metrics are averaged over these runs. We run the experiments with a window size of 3 elements for the convolution.

6.2.4 Results and Discussion

Learned rules and expressions Table 6.3 shows that the model converges towards accurate expressions. When looking at the learned rules,

they correspond to the ground truths (or logically equivalent expressions) in most cases. However we can point that for dataset 4, the model is not always reaching the ground truth. What differentiates dataset 4 from the others is that it involves the aggregation function sum. The codomain of aggregation function sum is wider than the ones for min, max and product. This might explain an increased sensitivity to the bias initialization and a possible need for more training time. In practice, the constrained threshold layer is responsible for this learning disparity, because the input value of this layer is not weighted by a trainable parameter, as it would be the case in a conventional linear layer.

Sparsity In terms of sparsity, the expressions, even when equivalent or very close to ground truths, are not always in their simplest form and might require simplification.

For example for dataset 4, the following expression was learned:

$$\exists s \text{ of size 3 in seq such that } \text{sum}_s\ F_5 > 2 \text{ and } \text{prod}_s\ F_5 > 0.2.$$

This could likely be solved per dataset with fine-tuning of hyper parameters or longer training, higher regularization coefficient λ or even enforced sparsity at the cost of finding an appropriate configuration.

In conclusion, this model, designed for learning rules with basic predefined aggregates on sequential data, has demonstrated its ability to generate expressive rules and great training performance. The user's choice of conv layer prevents this model from being suitable for any real use and it is therefore as expected only an intermediate model. However, it seems to be a good basis for integration into a more intricate and expressive model.

6.3 Convolutional Rule Neural Network with Aggregates (CR2NA)

In this section, we extend the s-CR2NA model by allowing for multiple convolutional filters to accommodate multiple sequential patterns types. We refer to this network as Convolutional Rule Neural Network with Aggregates (CR2NA).

6.3.1 A First Extension

6.3.1.1 Architecture

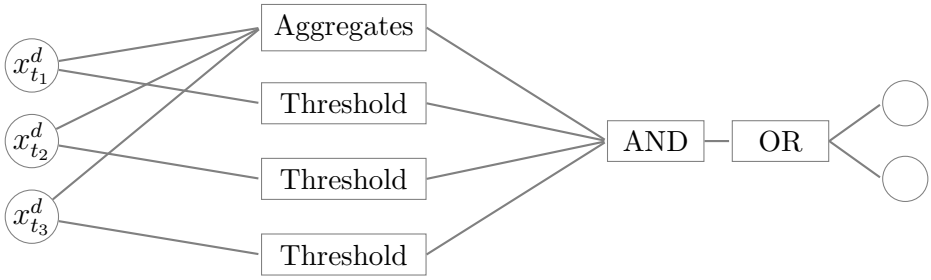


Figure 6.3: Example of filter proposed to learn rules with predefined aggregation functions on input $\boldsymbol{x}^d$ when used with the model presented in Figure 6.4. Illustrated is a filter with output size 2.

Filter The filter presented in Section 6.2.1 is modified slightly to support expressions based on raw numerical data. The Threshold layer is applied not only to the aggregated values, but also to raw numerical input. The resulting filter model is illustrated in Figure 6.3 and the grammar of one output neuron is described with production rules in Equation 6.5. Unlike previous architectures, the filter can have an output size higher than 1.

$$\begin{aligned}
\text{base expression} &\rightarrow \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\
\text{conjunction} &\rightarrow \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \\
\text{predicate} &\rightarrow \text{aggregate} \mid \text{literal} \\
\text{aggregate} &\rightarrow \text{aggfunction } \textit{of } \underline{x^d} \textit{ is greater than } \underline{\text{value}} \\
\text{aggfunction} &\rightarrow \textit{min} \mid \textit{max} \mid \textit{sum} \mid \textit{product} \\
\text{literal} &\rightarrow \underline{x^d} \textit{ is greater than } \underline{\text{value}} \textit{ at } \underline{t_i}
\end{aligned} \tag{6.5}$$

Conv Layers All four possible Conv Layers presented in Section 6.2.1 are applied to each output dimension of the filter. The layers are kept unchanged.

Rule Layers Finally all the resulting values from the different Conv Layers are given as input to the base rule model composed of an AND and an OR layer. Note that non sequential features could easily be added as input to the rule layers along with the resulting sequential patterns from the conv layers for datasets that are composed of both sequential and non sequential features.

An example of model with two dimensional filter is illustrated in Figure 6.4. Overall the rules represented by this model can be described by the production rules in Equation 6.6 (full grammar is available in Appendix C.6).

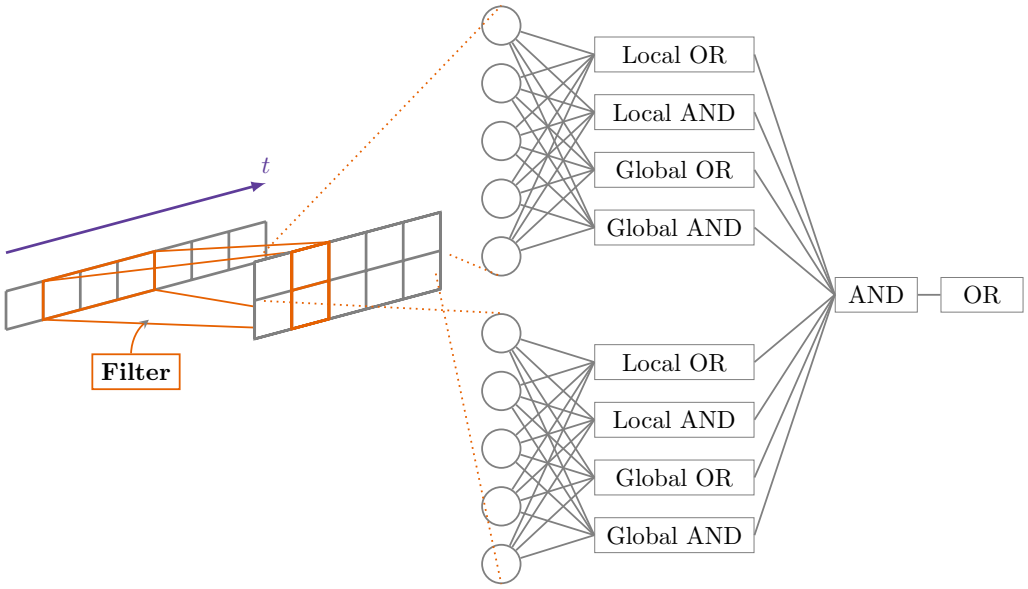


Figure 6.4: Base architecture of CR2NA with an example of a two dimensional filter.

$$\begin{array}{rcl} \text{rule} & \rightarrow & \textit{if}\ \text{disjunction}\ \textit{then}\ \textit{class} = 1\ \textit{else}\ \textit{class} = 0 \\ \text{disjunction} & \rightarrow & \text{conjunction} \mid \text{conjunction} \vee \text{disjunction} \\ \text{conjunction} & \rightarrow & \text{expr} \mid \text{expr} \wedge \text{conjunction} \\ \text{expr} & \rightarrow & \text{localOR} \mid \text{localAND} \mid \text{globalOR} \mid \text{globalAND} \end{array} \tag{6.6}$$

6.3.1.2 Training

The training strategy from Chapter 4 is kept unchanged (Section 4.2.2). The model parameters Θ of CR2NA are $\boldsymbol{W}_{and\_filter}, \boldsymbol{W}_{or\_filter}, \boldsymbol{W}_{conv}, \boldsymbol{W}_{and}, \boldsymbol{W}_{or}$ and $\boldsymbol{b}$.

Loss function Loss function from Equations 4.11 and 6.4 are still valid ($\lambda = 10^{-4}$). The regularization term Π, or *penalty*, still evaluates the number of terminal conditions in the rule. The number of terminal conditions of the base rule model Π_{base}, of a local model Π_{local} or of a global model Π_{global} can be expressed as functions of the model parameters as follows.

$$\boldsymbol{\Pi}_{base} = \boldsymbol{W}_{or\_filter}\boldsymbol{W}_{and\_filter}\mathbf{1}, \qquad \boldsymbol{\Pi}_{local} = \boldsymbol{\Pi}_{base} \odot (\boldsymbol{W}_{conv}\mathbf{1}), \qquad \boldsymbol{\Pi}_{global} = \boldsymbol{\Pi}_{base} \tag{6.7}$$

#	**Ground Truth**	**+class**	π
7	F_0 at $\mathtt{t}-5>0.7 \vee F_1$ at $\mathtt{t}-2>0.7$	51.8	2
8	F_1 at $\mathtt{t}-1>0.7 \vee \exists s$ *of size* 3 *in seq such that* $\min_s F_0 > 0.5$	60.7	2
9	$(F_1$ at $\mathtt{t}-1>0.5 \wedge F_8$ at $\mathtt{t}-1>0.6)$	55.5	4
	$\vee \quad (\sum_s F_4 > 1.5$ *over window:* s_{-1}		
	$\wedge \exists s$ *of size* 3 *in seq such that* $\min_s F_2 > 0.2)$		

Table 6.5: Ground truths applied on numerical input features $F_0, \ldots, F_9 \in [0,1]^{10}$ to generate 3 additional synthetic datasets along with the proportion of the positive class (in %) and the penalty π of the expression.

From which, Π can be expressed as follows.

$$\Pi(\Theta) = \boldsymbol{W}_{or}\boldsymbol{W}_{and}\begin{bmatrix}\boldsymbol{\Pi}_{localOR}\\ \boldsymbol{\Pi}_{localAND}\\ \boldsymbol{\Pi}_{globalOR}\\ \boldsymbol{\Pi}_{globalAND}\end{bmatrix} \tag{6.8}$$

6.3.1.3 Experiments

The validity and usefulness of this method is again evaluated by applying the model on synthetic datasets.

Synthetic Datasets In addition to the 6 synthetic datasets presented in Section 6.2.3, we propose 3 additional datasets in Table 6.5 with more complex rules and that involve the new aspects of the grammar supported by the model. There are 6000 sequences of 10 numerical features (F_i) ranging from 0 to 1. Sequences are of different lengths from 4 to 14 elements each. Generation is detailed in Appendix A.

Experimental Setting Experimental setting from previous section is still valid (Section 6.2.3) but the output size of the filter is set to 5 and the hidden size in the final rule model to 5.

6.3.1.4 Results and Discussion

When deepening the model and without radical changes in the training strategy, the training limitations of the model are exposed as shown in Table 6.6.

#	**Accuracy**	Π	**Epoch**
1	78.5 ± 7.7	5.2 ± 2.4	35.1 ± 10.4
2	85.3 ± 6.0	9.8 ± 2.8	101.4 ± 40.5
3	87.6 ± 4.3	6.9 ± 2.9	110.5 ± 24.7
4	86.0 ± 5.4	24.0 ± 7.9	81.7 ± 24.7
5	87.9 ± 4.3	27.2 ± 6.0	139.4 ± 41.2
6	80.6 ± 4.8	7.9 ± 2.6	102.2 ± 36.5
7	64.7 ± 3.8	8.7 ± 5.0	114.7 ± 27.6
8	68.8 ± 2.6	9.8 ± 4.2	189.0 ± 33.2
9	77.7 ± 4.6	7.0 ± 2.1	102.4 ± 19.6

Table 6.6: Average accuracies, penalties Π and best epochs (i.e. epoch of the best loss on validation set) obtained for the different datasets, along with the unbiased standard errors of the averages over the 10 executions with different weights initializations.

In a lot of cases, the model is learning the empty class which directly impacts the average and standard error values. There is one main reason to explain this training instability : the implementation of AND and OR layers combined with the latent weights. The AND and OR layers are implemented using the minimum function. It is applied to the weighted sum of the layer input which is by definition completely dependent on the layer input size (Equations 4.3 and 4.6). As a consequence, the larger the model, the greater the logical approximation errors during training. We believe this is the main cause of learning difficulties. Many other configurations have been tested in an attempt to find ideal parameters (learning rate scheduling, dropout, added noise,...). However, none of the configurations provided more stable and robust learning. This problem is all the more worrying as the datasets tested here are very simple and noiseless, hence the importance of addressing these learning limitations before possible usage on real-world data.

6.3.2 Improved Models

Following the observations of the previous model training difficulties, we experiment with other implementations of AND and OR layers to validate the unchanged overall CR2NA architecture.

6.3.2.1 Architectures

Every occurrence of an OR or AND layer in the previously introduced model (Section 6.3.1) is replaced with the following OR and AND layers. The computations in the new implementations have the benefit of providing meaningful approximation of the logical operation regardless of the input size. This was the main concern raised following the experiences with the previous implementation. These computations also come with a more meaningful approximations for input and weight values between 0 and 1. The rest of the architecture is kept unchanged.

Implementations are based on approximations of a common fuzzy logic OR operator : the maximum function.

The first set of OR and AND layers is based on the usage of the softmax function, they will be refereed to as the (soft)OR and (soft)AND layers.

(soft)OR Layer We exploit an approximation of the maximum with the softmax function to define an OR layer as follows ($*$ sizes are adjusted accordingly).

$$\boldsymbol{V} = n(\boldsymbol{W}^*_{or} \odot \boldsymbol{x}^*), \qquad \boldsymbol{y} = \text{diag}(\boldsymbol{V}^\intercal \text{softmax}(\boldsymbol{V})/n) \tag{6.9}$$

The softmax function corresponds to $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ with $z = (z_1, \ldots, z_K) \in \mathbb{R}^K$ applied on the input dimension K.

(soft)AND Layer In the same manner as in Section 4.2.1, applying the De Morgan's Law to Equation 6.9, we have:

$$\begin{aligned} \boldsymbol{V} &= n(\boldsymbol{W}^*_{and} \odot \neg\boldsymbol{x}^*), \qquad \boldsymbol{h} = \neg(\text{diag}(\boldsymbol{V}^\intercal \text{softmax}(\boldsymbol{V})/n)) \\ \text{i.e. } \boldsymbol{V} &= n(\boldsymbol{W}^*_{and} \odot (\boldsymbol{1} - \boldsymbol{x}^*)), \qquad \boldsymbol{h} = 1 - (\text{diag}(\boldsymbol{V}^\intercal \text{softmax}(\boldsymbol{V})/n)). \end{aligned} \tag{6.10}$$

The second set of OR and AND layers is based on the usage of the LogSumExp (LSE) function, and will be refered to as the (lse)OR and (lse)AND layers.

(lse)OR Layer We exploit an approximation of the maximum function with the LSE function to define a new OR layer as follows:

$$\boldsymbol{V} = n(\boldsymbol{W}^*_{or} \odot \boldsymbol{x}^*), \qquad \boldsymbol{s} = \text{logsumexp}(\boldsymbol{V})/n, \qquad \boldsymbol{y} = \min(\max(\boldsymbol{s}, 0), 1). \tag{6.11}$$

#	CR2N-soft Accuracy	CR2N-soft Π	CR2N-soft Epoch	CR2N-lse Accuracy	CR2N-lse Π	CR2N-lse Epoch
1	100.0 ± 0.0	8.8 ± 1.5	270.2 ± 30.3	100.0 ± 0.0	11.5 ± 1.6	32.6 ± 2.9
2	99.5 ± 0.5	11.1 ± 1.7	238.7 ± 23.2	100.0 ± 0.0	19.4 ± 1.9	56.4 ± 5.1
3	100.0 ± 0.0	25.4 ± 4.3	82.3 ± 15.6	99.8 ± 0.2	15.9 ± 2.1	31.5 ± 7.4
4	99.3 ± 0.7	12.0 ± 1.6	99.4 ± 12.4	100.0 ± 0.0	15.1 ± 2.0	49.7 ± 5.0
5	96.7 ± 0.8	13.3 ± 2.2	142.3 ± 46.5	98.0 ± 0.6	14.9 ± 2.3	48.8 ± 5.0
6	100.0 ± 0.0	5.4 ± 0.9	302.8 ± 27.5	97.1 ± 1.0	7.9 ± 1.0	59.6 ± 12.1
7	85.2 ± 1.9	**40.0 ± 6.2**	276.9 ± 38.4	78.6 ± 0.8	14.3 ± 1.9	87.6 ± 17.6
8	82.2 ± 2.4	**51.6 ± 5.5**	263.9 ± 35.7	76.7 ± 1.3	16.9 ± 3.0	93.8 ± 23.1
9	86.7 ± 0.6	22.9 ± 4.9	225.0 ± 33.6	87.5 ± 0.3	13.2 ± 2.8	90.3 ± 19.4

Table 6.7: Average accuracies, penalties Π and best epochs (i.e. epoch of the best loss on validation set) obtained for the different datasets, along with the unbiased standard errors of the averages over the 10 executions with different weights initializations.

LSE function corresponds to $\text{logsumexp}(z_i) = \log \sum_j e^{z_{i,j}}$ and is applied on the input dimension.

(lse)AND Layer Applying the De Morgan's Law to Equation 6.11, we have:

$$\begin{aligned} \boldsymbol{V} &= n(\boldsymbol{W}^*_{and} \odot \neg \boldsymbol{x}^*), & \boldsymbol{h} &= \neg \min(\max(\boldsymbol{s}, 0), 1) \\ \text{i.e. } \boldsymbol{V} &= n(\boldsymbol{W}^*_{and} \odot (\boldsymbol{1} - \boldsymbol{x}^*)), & \boldsymbol{h} &= 1 - \min(\max(\boldsymbol{s}, 0), 1). \end{aligned} \tag{6.12}$$

6.3.2.2 Training

Training is kept the same as described in Section 6.3.1.2.

6.3.2.3 Experiments

We will refer to the former model described in Section 6.3.1 as CR2NA model, the one based on (soft)AND and OR layers as the CR2NA with Softmax implementation (CR2NA-soft) and the one built on top of (lse) layers as CR2NA with LSE implementation (CR2NA-lse). In order to compare the three, the experimental setting is kept the same as in Section 6.3.1.3.

6.3.2.4 Results and Discussion

The results obtained for the learning of CR2NA-soft and CR2NA-lse are presented in Table 6.7. First of all in terms of accuracy, performances are comparable for simplest datasets but CR2NA-soft reaches better accuracies than CR2NA-lse on the 3 most complex datasets (datasets 7, 8 and 9). It is very likely related to the difference in speed of convergence of the two implementations. The average best epoch for CR2NA-soft is much higher than for CR2NA-lse. It shows that only a fourth of the training time in terms of epochs is used by CR2NA-lse model to converge towards the best model. Note that these results were obtained without specific fine tuning and with a basic training strategy, altough these results suggest that CR2NA-lse could benefit from smaller learning rate or learning rate scheduling for instance.

In terms of penalties, both models obtained comparable penalties for simplest models (1 to 6) that are quite high compared to ideal π values presented in Table 6.2 and 6.5 but also compared to the ones obtained for CR2NA (Table 6.6). Stronger regularization (dropout or added noise can also be considered), longer training or enforced sparsity (at the cost of finding an appropriate configuration) would be required for improvement of the learned rules complexity and overall generalization performance for complex datasets compared to the training of CR2NA (at least on these simple datasets).

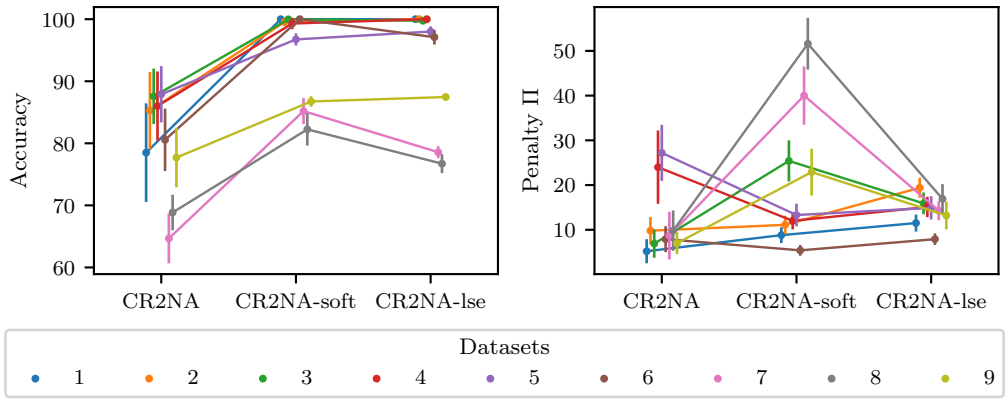


Figure 6.5: Average accuracies and penalties Π obtained for the different datasets per model, along with the standard errors of the average over the 10 executions with different weights initialization.

When compared to CR2NA original implementation (Section 6.3.1) the training is much more consistent. This is highlighted in Figure 6.5, with CR2NA-soft and CR2NA-lse implementations improvements in terms of accuracy for all datasets but also with the very low standard errors for both

accuracy and penalty values. Average penalty being 0 when the CR2NA model converges towards the empty rule makes the average penalty comparison with this model not particularly relevant.

In terms of average training time of an epoch, CR2NA and CR2NA-lse require comparable time but are two times faster than CR2N-soft which suggest that CR2NA-lse implementation could be a preferred in practice.

Also, although it was not required with these synthetic datasets, for more complex datasets we suggest to avoid the addition of noise in the computation of the threshold layer as well as Dropout in the conv layers as common regularization technique to learn less represented patterns in the data.

To conclude, this deeper architecture, designed for learning rules with predefined aggregates on sequential data, has demonstrated its ability to generate expressive rules and great training performance with specific base layers implementations. Indeed with the base AND and OR layers described in Chapter 4, the experiments highlighted some training limitations of their implementation. Following this observation, we investigated other logical layers implementations to address this issue based on approximation of the maximum function (with softmax and LSE functions). Both offer a more stable training but come with resulting rules with higher penalty values. A direction for future work would be to investigate how it could be improved efficiently with the addition of a enforced sparsity strategy, more training time, higher regularization coefficient or hyper parameters tuning and scheduling for example. Implementations based on LSE seems to provide the best convergence and computation time.

6.4 Conclusion

To conclude, with the approach of deepening the base rule model we were able to extend the rule grammar and expressivity of the learned rules. First a simple intermediate model applying predefined aggregation functions to sequential data was trained and tested as a proof-of-concept to encourage this research direction. Then the architecture was extended with additional rule layers that extend the rules complexity to another level with the possibility to learn rules based on multiple sequential filters (and non sequential features). Increasing the size of the model did, however, highlight learning limitations of the original implementations of the AND and OR layers. Other implementations were explored and compared when used in the same model architecture and proved to provide more stable training and consistent performance across experiments.

This architecture is very promising and future work will consist in investigating its limitations with non-perfect data and existing datasets. These results pave the way for many possible improvements in terms of expressiveness and learning.

Chapter 7

Conclusion

We started from the observation that, to the best of our knowledge, there was no existing method to learn expressive rules, i.e. rules with a grammar that can deal with different types of input data and express complex expressions. If we consider the full learning process to take as input the raw input data and return as output the classification rules, there are many specialized approaches that focus on a specific step in this process but none target the overall goal. Betting on an end-to-end neural-based approach, we propose such models starting from a base rule model that learns a disjunction of conjunctions (Chapter 4) and step by step extending it to more complex grammars (Chapter 5 and 6).

Contributions In the path of extending the grammars of learned classification rules, we started with a preliminary work based on a preprocessing approach. Then, on the basis of a neural-based rule learning approach for binary tabular data we provided support for numerical and categorical data, compared the learning of DNF to ANF expressions and discussed extensions for other classification problems (multi-classification and multi-label classification). Two models were proposed for supporting sequential data (RR2N and CR2N) before the addition of predefined aggregates to the rule grammar together with a deeper architecture (CR2NA). This final architecture is much more expressive and closer to what a business user can write in a BRMS than the first RN model or state-of-art rule learning models. The main contributions of this thesis are:

- CR2N, a convolutional rule neural network which learns different types of sequential patterns alongside the classification rules.
- CR2NA, an expressive rule neural network that can express rules with predefined temporal aggregates but also, due to its flexible architecture,

it can deal with both sequential and flat data of different types : binary, numerical and categorical data.

Limitations The work presented here does come with important limitations. First of all, we focused the work on binary classification problems for simplicity and it restrained quite significantly the existing datasets we could use and thus the possibilities for us to compare our model to other approaches. As a consequence, experiments are quite limited and almost exclusively conducted using synthetic data. Experiments on synthetic data are required to validate the model as a first step but most certainly are not enough to show all of their limitations. During the writing of this manuscript, RL-Net model proposed a very interesting solution to deal with multi classification problems [Dierckx et al., 2023]. It would be of great interest to incorporate their ideas in our models to evaluate them in a broader scope.

Then, throughout the entire manuscript, we motivated the application of learned rules with fraud detection in mind. Fraud detection problems are characterized by a distinct feature: highly unbalanced data. We made the deliberate decision not to go in that direction and instead decided to keep the proposed methods as generic as possible. However, the proposed architectures are not ready to use without prior data processing (e.g. under or over sampling) in highly unbalanced settings. Also, the main motivation of this work is related to BRMS. However, the commutativity properties of the logical operations in use are fully exploited and thus obtained expressions are unordered. In business rules the order of conditions (and rules) can have a huge importance for error management.

Finally, looking more closely into the technical limitations of our proposed approaches, we are using a tool, neural networks, and an optimization strategy, gradient descent, that is not specifically designed for discrete optimization problems such as ours. This means that in addition to using tips and tricks to make it to work (latent weights, enforced sparsity for example) the well-known limitations of neural networks also apply and some might even be exacerbated. We are referring to sensitivity to hyperparameters, difficulties in optimizing non-linear functions, vanishing and/or exploding gradients or sensitivity to initialization for example. However to some extent (length of the rule can still impact interpretability) lack of interpretability is no longer a problem.

Perspectives for future work In addition to addressing these limitations, there are still other avenues for future work. First, there are still a lot of complex expressions both for flat and sequential data that should be learnable

in order to match BRMS supported rule grammar. In this direction itself, there is still much to be done. For example, we can mention sequential operators like *before*, *after* or *occurs in* (looking into Linear Temporal Logic (LTL) would be a great starting point). Also the support for sequential data could be extended to time series with some well-chosen filter size and support for ordered categorical data could be investigated. It is also important to note that BRMS are also not just used for binary classification problems. Extending the expressivity of the right side of rules (action part) could also be of great interest and require further exploration within the research domains of regression problems and/or program synthesis.

Another direction of research would be to look into the stability and robustness of the training strategy. We consider the presented work to provide initial steps in the direction of a working end-to-end rule learning process rather than a final solution. Further studies on weights initialization, optimization of BNN and layers implementations would likely improve the quality of the training for instance.

On the longer term, we believe it would be interesting to investigate the use of this approach for another type of data we have not mentioned yet: graphs. There are many different machine learning problems that could be explored with a rule-based approach such as node classification, edge prediction or graph clustering. We have seen CNN being used on graph data in the past [Defferrard et al., 2017], so why not consider a CNN with an interpretable filter as presented in this manuscript.

Finally, if we believe that interpretable AI will follow the steps of generic AI, there are still many existing tools and techniques to be adapted, optimized or used as inspiration for interpretable neural models. From Hidden Markov Model (HMM), to RNN, to CNN, to transformers and very recently Large Language Models (LLMs), AI has experienced remarkable progress and development in recent years most of which are yet to be made interpretable. This promises a very bright future for neural-based rule learning if for example interpretable transformers are to exist. There is currently a considerable research (and industrial) interest in LLMs. These models are language models which, until now, are not suitable for learning mathematical expressions (and thus rules) or for arithmetic reasoning. However, in the context of business rules and BRMS, they are extremely relevant for proposing rules from a textual query or for assisting the writing of rules for business users (as an AI pair programmer). An hybrid approach combining the power of LLM and a neural-based rule learning model would be a very interesting avenue to pursue as well.

Another widely used learning technique for non interpretable models that is not commonly applied to rule learning is reinforcement learning. Adapting

the current approach with reinforcement learning would be another direction for future work. This would likely benefit interactive real-world environment although this would not necessarily be in the immediate scope of business rules. It might also more generally bring robustness to the rule construction. This year, reinforcement learning was introduced for the learning of temporal rules in a different set up [Yang et al., 2023] and showed promising results.

Appendices

A Benchmarks proposal

In this section we gather all the information relative to the new synthetic datasets proposed to evaluate binary classification rule learning models.

We refer to the set of datasets presented in Table 1 as the Binary Classification Rule Dataset (BCRD) collection (Available here : `https://github.com/IBM/classification-rule-datasets`).

Synthetic datasets generation Datasets are created by applying ground truths on randomly generated features.

For the balanced datasets in use in Chapter 5, they are generated randomly with the same ground truth as unbalanced datasets. Then, they are upsampled until the minority class represents half of the goal dataset size and appropriate number of majority class are randomly removed.

#	**Ground Truth**	**+class**	π	π^*_{DNF}
sl	*Simple Logical Rules*			
	10000 instances composed of 10 binary features $F_0, \ldots, F_9 \in \{0,1\}^{10}$			
1	$F_0 \vee (F_3 \wedge F_6)$	63.5	3	
2	$(F_0 \wedge F_4) \vee (F_3 \wedge F_6)$	43.6	4	
3	$(\neg F_3 \wedge \neg F_6) \vee \neg F_0 \vee \neg F_4$	80.6	4	
4	$(\neg F_3 \wedge \neg F_6) \vee (F_0 \wedge F_9 \wedge \neg F_6) \vee \neg F_0 \vee \neg F_4$	83.9	7	6
5	$F_0 \veebar F_4$	49.6	2	4
6	$F_0 \veebar F_4 \veebar F_8$	49.9	3	12
7	$F_0 \veebar F_4 \veebar (F_3 \wedge F_8)$	49.5	4	20
8	$F_0 \veebar F_4 \veebar (F_8 \wedge \neg F_3)$	50.2	4	20
st	*Short-term dependency Rules on Sequences*			
	10000 sequences of length 7 composed of 10 binary features $F_0, \ldots, F_9 \in \{0,1\}^{10}$			
1	F_1 at t-1 $\wedge$ F_2 at t	25.0	2	
2	F_1 at t-1 $\vee$ F_2 at t	75.1	2	
3	F_4 at t-1	50.5	1	
4	F_4 at t-2	50.0	1	
5	F_4 at t-3	49.7	1	
6	F_4 at t-4	49.6	1	
7	F_4 at t-5	50.0	1	
lg	*Rules with Local and Global patterns on Sequences*			
	1000 sequences of length from 4 to 14 composed of letters (A to F)			
1	C at t-4	14.2	1	
2	A at t-6 $\wedge$ C at t-4	1.5	2	
3	(A at t-6 $\wedge$ C at t-4) $\vee$ (B at t-5 and C at t-3)	3.6	4	
4	B-D in sequence	20.4	2	
sa	*Rules with Aggregates*			
	6000 sequences of length from 4 to 14 composed of 10 numerical features $F_0, \ldots, F_9 \in [0,1]^{10}$			
1	$\max_s F_0 > 0.8$ *over window:* s_{-1}	49.2	1	
2	$\max_s F_0 > 0.9$ *over any of the windows:* s_{-1}, s_{-4}	43.9	2	
3	$\max_s F_0 > 0.9$ *over all of the windows:* s_{-1}, s_{-4}	62.7	2	
4	$\exists s$ *of size* 3 *in seq such that* $\sum_s F_5 > 2$	55.0	1	
5	$\exists s$ *of size* 3 *in seq such that* $\prod_s F_5 > 0.25$	52.8	1	
6	$\forall s$ *of size* 3 *in seq,* $\min_s F_0 > 0.05$	64.5	1	
7	F_0 at t $- 5 > 0.7 \vee F_1$ at t $- 2 > 0.7$	51.8	2	
8	F_1 at t $- 1 > 0.7 \vee \exists s$ *of size* 3 *in seq such that* $\min_s F_0 > 0.5$	60.7	2	
9	(F_1 at t $- 1 > 0.5 \wedge F_8$ at t $- 1 > 0.6$) $\vee$ ($\sum_s F_4 > 1.5$ *over window:* s_{-1} $\wedge \exists s$ *of size* 3 *in seq such that* $\min_s F_2 > 0.2$)	55.5	4	

Table 1: Ground truths applied on specified input features to generate the BCRDs along with the proportion of the positive class (in %), π, the number of conditions in the expression, π^*_{DNF} the number of conditions in the simplified equivalent DNF expression when different from π.

B Manual unit labeling of UCI datasets

Manual unit labeling of UCI datasets required for experiments presented in Chapter 3 is described in Table 2.

Dataset	Unit	Features
abalone	gr	whole, shucked, viscera, shell
	mm	length, diam, height
adult	u_1	age
	u_2	fnlwgt
	u_3	education num
	u_4	capital gain, capital loss
	u_5	hours week
breast wisconsin	u_1	clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses
	index	sample code number
iris	cm	sepal length, sepal width, petal length, petal width
segment	u_1	region centroid col, region centroid row
	u_2	region pixel count
	u_3	short line density 5, short line density 2
	u_4	vedge mean, hedge mean, intensity mean, rawred mean, rawblue mean, rawgreen mean
	u_5	vedge sd, hedge sd
	u_6	exred mean, exblue mean, exgreen mean
	u_7	value mean, saturatoin mean, hue mean
wine	u_1	alcohol, malic acid, ash, alcalinity of ash, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, od280/od315
	u_2	magnesium, proline
wine quality	u_1	fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, sulphates
	u_2	density
	u_3	ph
	u_4	alcohol

Table 2: Unit decomposition for all numerical features of selected UCI datasets. u_k refers to the k^{th} unnamed unit of a dataset.

C Context-Free Grammars

C.1 RN

The rule language of RN model (Section 4.2) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{array}{lcl}
V_T & = & \{\textit{if}, \textit{then class = 1 else class = 0}\} \cup \{\wedge, \vee\} \cup \left\{\underline{x_1}, \underline{x_2}, \dots\right\} \\
V_N & = & \{\text{rule}, \text{base expression}, \text{conjunction}, \text{predicate}\} \\
S & = & \{\text{rule}\} \\
R & = & \left\{\begin{array}{rcl}
\text{rule} & \to & \textit{if}\ \text{base expression}\ \textit{then class = 1 else class = 0} \\
\text{base expression} & \to & \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\
\text{conjunction} & \to & \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \\
\text{predicate} & \to & \underline{x_1} \mid \underline{x_2} \mid \dots
\end{array}\right.
\end{array}
$$

C.2 RN-anf

The rule language of RN-anf model (Section 4.3) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{array}{lcl}
V_T & = & \{\textit{if}, \textit{then class = 1 else class = 0}\} \cup \{\wedge, \veebar, \textit{True}\} \cup \left\{\underline{x_1}, \underline{x_2}, \dots\right\} \\
V_N & = & \{\text{rule}, \text{base expression}, \text{conjunction}, \text{predicate}\} \\
S & = & \{\text{rule}\} \\
R & = & \left\{\begin{array}{rcl}
\text{rule} & \to & \textit{if}\ \text{base expression}\ \textit{then class = 1 else class = 0} \\
\text{base expression} & \to & \text{conjunction} \mid \text{conjunction} \veebar \text{base expression} \\
\text{conjunction} & \to & \text{predicate} \mid \text{predicate} \wedge \text{conjunction} \mid \textit{True} \\
\text{predicate} & \to & \underline{x_1} \mid \underline{x_2} \mid \dots
\end{array}\right.
\end{array}
$$

C.3 RR2N

The rule language of RR2N model (Section 5.2) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{array}{lcl}
V_T & = & \{\textit{if}, \textit{then class = 1 else class = 0}, \textit{at}\ \texttt{t}-, \} \cup \{\wedge, \vee\} \cup \left\{\underline{x^d}, \underline{t}\right\} \\
V_N & = & \{\text{rule}, \text{base expression}, \text{conjunction}_t, \text{literal}_t\} \\
S & = & \{\text{rule}\} \\
R & = & \left\{\begin{array}{rcl}
\text{rule} & \to & \textit{if}\ \text{base expression}\ \textit{then class = 1 else class = 0} \\
\text{base expression} & \to & \text{conjunction}_t \mid \text{conjunction}_t \vee \text{base expression} \\
\text{conjunction}_t & \to & \text{literal}_t \mid \text{conjunction}_{t-1} \mid \text{literal}_t \wedge \text{conjunction}_t \\
 & & \mid \text{conjunction}_{t-1} \wedge \text{conjunction}_t \\
\text{literal}_t & \to & \underline{x^d}\ \textit{at}\ \texttt{t}-\underline{t}
\end{array}\right.
\end{array}
$$

C.4 CR2N

The rule language of CR2N model (Section 5.3) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{aligned}
V_T &= \{\textit{if}, \textit{then class = 1 else class = 0}, \textit{at } \texttt{t}-, \} \cup \{\wedge, \vee, *, -\} \cup \\
&\quad \{\underline{x_1}, \underline{x_2}, \dots\} \cup \left\{\underline{x_{c_1} = \alpha^1_0}, \dots, \underline{x_{c_1} = \alpha^1_{n_1}}, \dots, \underline{x_{c_k} = \alpha^k_{n_k}}\right\} \cup \{\underline{i}\} \\
V_N &= \{\text{rule}, \text{expression}, \text{local pattern}, \text{global pattern}, \text{base expression}_{local}, \\
&\quad \text{base expression}_{global}, \text{conjunction}_{local}, \text{conjunction}_{global}, \text{predicate}_{local}, \\
&\quad \text{predicate}_{global}, \text{categorical expression}, \text{categorical literal}, \text{literal}\} \\
S &= \{\text{rule}\}
\end{aligned}
$$

$$
R = \left\{
\begin{array}{rcl}
\text{rule} & \rightarrow & \textit{if } \text{expression } \textit{then class} = 1 \textit{ else class} = 0 \\
\text{expression} & \rightarrow & \text{local pattern} \mid \text{global pattern} \\
\text{local pattern} & \rightarrow & \text{base expression}_{local} \\
\text{global pattern} & \rightarrow & \text{base expression}_{global} \\
\\
\text{base expression}_{local} & \rightarrow & \text{conjunction}_{local} \mid \\
 & & \text{conjunction}_{local} \vee \text{base expression}_{local} \\
\text{conjunction}_{local} & \rightarrow & \text{predicate}_{local} \mid \text{predicate}_{local} \wedge \text{conjunction}_{local} \\
\text{predicate}_{local} & \rightarrow & \text{categorical expression } \textit{at } \texttt{t} - \underline{i} \mid \text{literal } \textit{at } \texttt{t} - \underline{i}. \\
\\
\text{base expression}_{global} & \rightarrow & \text{conjunction}_{global} \mid \\
 & & \text{conjunction}_{global} \vee \text{base expression}_{global} \\
\text{conjunction}_{global} & \rightarrow & \text{predicate}_{global} \mid * \mid \\
 & & \text{predicate}_{global} - \text{conjunction}_{global} \\
\text{predicate}_{global} & \rightarrow & \text{categorical expression} \mid \text{literal} \\
\\
\text{categorical expression} & \rightarrow & \text{categorical literal} \mid \\
 & & \text{categorical literal} \vee \text{categorical expression} \\
\text{categorical literal} & \rightarrow & \underline{x_c = \alpha^c_0} \mid \dots \mid \underline{x_c = \alpha^c_{n_c}} \\
 & & (\text{or simply } \underline{\alpha^c_0} \mid \dots \mid \underline{\alpha^c_{n_c}}) \\
\text{literal} & \rightarrow & \underline{x_1} \mid \underline{x_2} \mid \dots
\end{array}
\right.
$$

C.5 s-CR2NA

The rule language of s-CR2NA model (Section 6.2) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{aligned}
V_T &= \{\textit{if}, \textit{then class = 1 else class = 0}\} \cup \{\textit{over any of the windows}, \textit{over all of the windows}, \exists, \forall, \textit{of size}, \textit{in seq}, \textit{such that}, \textit{of}, \textit{is greater than}\} \cup \{\wedge, \vee\} \cup \left\{\underline{x^d}, \underline{\text{value}}, \underline{\hat{S}}, \underline{S}, \underline{L}\right\} \cup \{\textit{min}, \textit{max}, \textit{sum}, \textit{product}\} \\
V_N &= \{\text{rule}_{\textit{localOR}}, \text{rule}_{\textit{localAND}}, \text{rule}_{\textit{globalOR}}, \text{rule}_{\textit{globalAND}}, \text{localOR}, \text{localAND}, \text{globalOR}, \text{globalAND}, \text{base expression}, \text{conjunction}, \text{aggregate}, \text{aggfunction}\} \\
S &= \{\text{rule}_{\textit{localOR}}, \text{rule}_{\textit{localAND}}, \text{rule}_{\textit{globalOR}}, \text{rule}_{\textit{globalAND}}\} \\
R &= \begin{cases}
\text{rule}_{\textit{localOR}} \rightarrow \textit{if}\ \text{localOR}\ \textit{then class} = 1\ \textit{else class} = 0 \\
\text{rule}_{\textit{localAND}} \rightarrow \textit{if}\ \text{localAND}\ \textit{then class} = 1\ \textit{else class} = 0 \\
\text{rule}_{\textit{globalOR}} \rightarrow \textit{if}\ \text{globalOR}\ \textit{then class} = 1\ \textit{else class} = 0 \\
\text{rule}_{\textit{globalAND}} \rightarrow \textit{if}\ \text{globalAND}\ \textit{then class} = 1\ \textit{else class} = 0 \\
\\
\text{localOR} \rightarrow \text{base expression}\ \textit{over any of the windows:}\ \underline{\hat{S}} \\
\text{localAND} \rightarrow \text{base expression}\ \textit{over all of the windows:}\ \underline{\hat{S}} \\
\text{globalOR} \rightarrow \exists \underline{S}\ \textit{of size}\ \underline{L}\ \textit{in seq such that}\ \text{base expression} \\
\text{globalAND} \rightarrow \forall \underline{S}\ \textit{of size}\ \underline{L}\ \textit{in seq,}\ \text{base expression} \\
\\
\text{base expression} \rightarrow \text{conjunction} \mid \text{conjunction} \vee \text{base expression} \\
\text{conjunction} \rightarrow \text{aggregate} \mid \text{aggregate} \wedge \text{conjunction} \\
\text{aggregate} \rightarrow \text{aggfunction}\ \textit{of}\ \underline{x^d}\ \textit{is greater than}\ \underline{\text{value}} \\
\text{aggfunction} \rightarrow \textit{min} \mid \textit{max} \mid \textit{sum} \mid \textit{product}
\end{cases}
\end{aligned}
$$

C.6 CR2NA

The rule language of CR2NA model (Section 6.3) is described by the context-free grammar $G = (V_T, V_N, S, R)$ (as defined in Section 1.1.2) with:

$$
\begin{aligned}
V_T &= \{\textit{if}, \textit{then class = 1 else class = 0}\} \cup \{\textit{over any of the windows}, \\
&\quad \textit{over all of the windows}, \exists, \forall, \textit{of size}, \textit{in seq}, \textit{such that}, \textit{of}, \\
&\quad \textit{is greater than}, \textit{at}\} \cup \{\wedge, \vee\} \cup \left\{\underline{x^d}, \underline{\text{value}}, \underline{\hat{S}}, \underline{S}, \underline{L}, \underline{c}, \underline{t_i}\right\} \cup \{\textit{min}, \textit{max}, \textit{sum}, \\
&\quad \textit{product}\} \\
V_N &= \{\text{rule, disjunction, conjunction, expr, localOR, localAND, globalOR, globalAND, base expression, conjunction}_{filter}\text{, predicate, aggregate, aggfunction, literal}\} \\
S &= \{\text{rule}\} \\
R &= \begin{cases}
\text{rule} \rightarrow \textit{if}\ \text{disjunction}\ \textit{then class} = 1\ \textit{else class} = 0 \\
\text{disjunction} \rightarrow \text{conjunction} \mid \text{conjunction} \vee \text{disjunction} \\
\text{conjunction} \rightarrow \text{expr} \mid \text{expr} \wedge \text{conjunction} \\
\text{expr} \rightarrow \text{localOR} \mid \text{localAND} \mid \text{globalOR} \mid \text{globalAND} \\
\\
\text{localOR} \rightarrow \text{base expression}\ \textit{over any of the windows:}\ \hat{S} \\
\text{localAND} \rightarrow \text{base expression}\ \textit{over all of the windows:}\ \hat{S} \\
\text{globalOR} \rightarrow \exists S\ \textit{of size}\ L\ \textit{in seq such that}\ \text{base expression} \\
\text{globalAND} \rightarrow \forall S\ \textit{of size}\ L\ \textit{in seq,}\ \text{base expression} \\
\\
\text{base expression} \rightarrow \text{conjunction}_{filter} \mid \text{conjunction}_{filter} \vee \text{base expression} \\
\text{conjunction}_{filter} \rightarrow \text{predicate} \mid \text{predicate} \wedge \text{conjunction}_{filter} \\
\text{predicate} \rightarrow \text{aggregate} \mid \text{literal} \\
\text{aggregate} \rightarrow \text{aggfunction}\ \textit{of}\ \underline{x_d}\ \textit{is greater than}\ \underline{\text{c}} \\
\text{aggfunction} \rightarrow \textit{min} \mid \textit{max} \mid \textit{sum} \mid \textit{product} \\
\text{literal} \rightarrow \underline{x_d}\ \textit{is greater than}\ \underline{\text{value}}\ \textit{at}\ \underline{t_i}
\end{cases}
\end{aligned}
$$

D Peptides Dataset

UCI anticancer peptides dataset [Grisoni et al., 2019] (Available on [Dua and Graff, 2017]) is composed of one-letter amino acid sequences (of variable length) and each sequence is labeled with its anticancer activity on breast cancer cell lines.

The dataset provides 4 classes with the following distribution: 83 inactive-exp, 750 inactive-virtual, 98 moderately active and 18 very active. Sequences lengths range from 5 to 38 letters (Mean: 17 ± 5.5).

We transform this multi-classification problem into a binary classification problem (as done in [Nwegbu et al., 2022]). Class 'inactive-virtual' is the positive class (750) and all the other are combined as the negative class (199). No other processing of the data is necessary and we leave it as is.

E Experimental Setting

The $\boldsymbol{C}$ latent weights are initialized with xavier uniform initialization method [Glorot and Bengio, 2010].

Experiments were run on CPU on a MacBookPro18,2 (2021) with Apple M1 Max chip, 10 Cores, 32 GB of RAM and running macOS Monterey Version 12.4.

F Exploded views of the Recurrent Rule Neural Network (RR2N)

Two exploded views of the example of trained RR2N model from Figure 5.1 are shown in Figure 1 and 2. They are equivalent to the representation in Figure 5.1 and illustrate the recurrence of the layer by unfolding it in the visual representation.

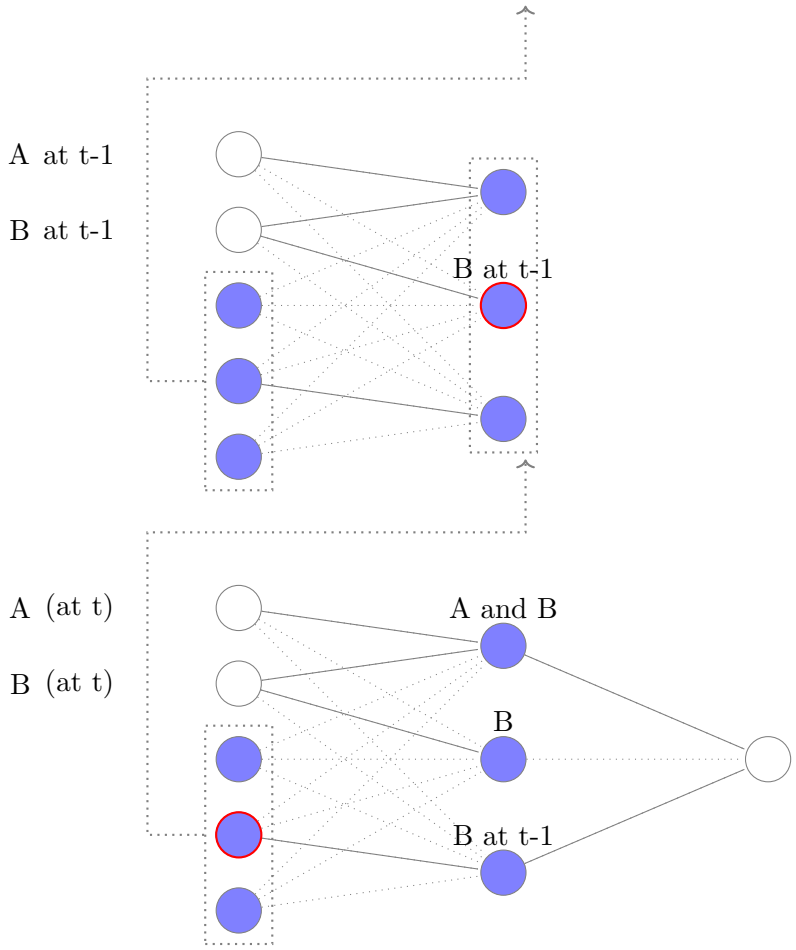


Figure 1: Exploded view of RR2N model.

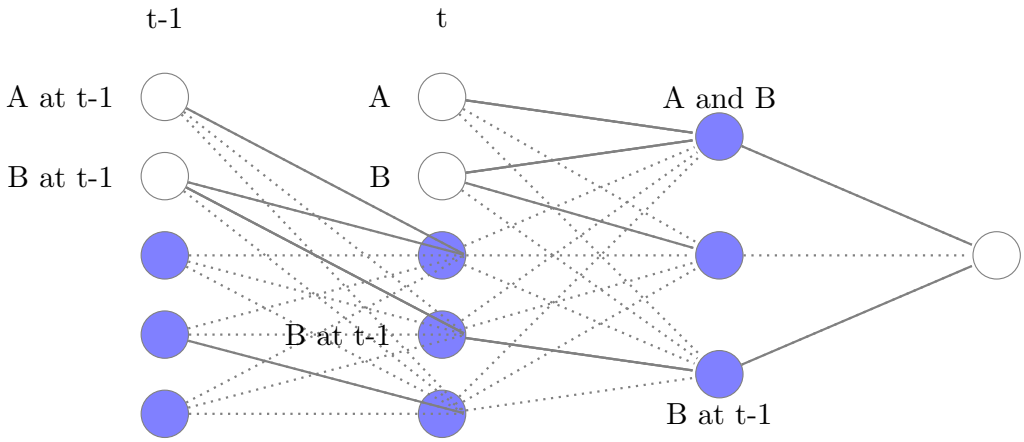


Figure 2: Flat view of RR2N model.

Bibliography

[IBM, 2021] (2021). IBM ODM Documentation. www.ibm.com/docs/en/odm.

[Aggarwal, 2002] Aggarwal, C. C. (2002). On effective classification of strings with wavelets. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 163–172, New York, NY, USA. Association for Computing Machinery.

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Alnababteh et al., 2014] Alnababteh, M., Alfyoumi, M., Aljumah, A., and Ababneh, J. (2014). Associative Classification Based On Incremental Mining (ACIM). *International Journal of Computer Theory and Engineering(IJCTE*, 6:135–140.

[Aloni, 2023] Aloni, M. (2023). Disjunction. In Zalta, E. N. and Nodelman, U., editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2023 edition.

[Arnold et al., 2005] Arnold, K., Gosling, J., and Holmes, D. (2005). *The Java Programming Language*. Addison Wesley Professional.

[Barbiero et al., 2023] Barbiero, P., Ciravegna, G., Giannini, F., Zarlenga, M. E., Magister, L. C., Tonda, A., Lio', P., Precioso, F., Jamnik, M., and Marra, G. (2023). Interpretable neural-symbolic concept reasoning.

[Beck and Fürnkranz, 2021a] Beck, F. and Fürnkranz, J. (2021a). Beyond DNF: First Steps towards Deep Rule Learning. In Brejová, B., Ciencialová, L., Holeňa, M., Mráz, F., Pardubská, D., Plátek, M., and Vinař, T., editors, *Proceedings of the 21st Conference Information Technologies –*

Applications and Theory (ITAT 2021), volume 2962 of *CEUR Workshop Proceedings*, pages 61–68, Hotel Heľpa, Nízke Tatry and Muránska planina. CEUR.

[Beck and Fürnkranz, 2021b] Beck, F. and Fürnkranz, J. (2021b). An Empirical Investigation Into Deep and Shallow Rule Learning. *Frontiers in Artificial Intelligence*, 4.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[Bennett and Mangasarian, 1992] Bennett, K. P. and Mangasarian, O. L. (1992). *Robust Linear Programming Discrimination Of Two Linearly Inseparable Sets*.

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. Taylor & Francis.

[Brudermueller et al., 2020] Brudermueller, T., Shung, D. L., Stanley, A. J., Stegmaier, J., and Krishnaswamy, S. (2020). Making Logic Learnable With Neural Networks.

[Cameron et al., 2020] Cameron, C., Chen, R., Hartford, J., and Leyton-Brown, K. (2020). Predicting Propositional Satisfiability via End-to-End Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3324–3331.

[Cherrier, 2021] Cherrier, N. (2021). *Interpretable Machine Learning for CLAS12 Data Analysis*. PhD thesis, Université Paris-Saclay.

[Chomsky, 1956] Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283.

[Cohen, 1995] Cohen, W. W. (1995). Fast Effective Rule Induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.

[Collery et al., 2023] Collery, M., Bonnard, P., Fages, F., and Kusters, R. (2023). Neural-based classification rule learning for sequential data. In *International Conference on Learning Representations*.

[Cortez et al., 2009] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553.

[d'Avila Garcez et al., 2019] d'Avila Garcez, A., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. (2019). Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning.

[d'Avila Garcez and Lamb, 2023] d'Avila Garcez, A. and Lamb, L. C. (2023). Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12406.

[De Giacomo et al., 2022] De Giacomo, G., Favorito, M., Li, J., Vardi, M. Y., Xiao, S., and Zhu, S. (2022). LTLf Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 2591–2598, Vienna, Austria. International Joint Conferences on Artificial Intelligence Organization.

[Defferrard et al., 2017] Defferrard, M., Bresson, X., and Vandergheynst, P. (2017). Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.

[Dierckx et al., 2023] Dierckx, L., Veroneze, R., and Nijssen, S. (2023). RL-Net: Interpretable Rule Learning with Neural Networks. In Kashima, H., Ide, T., and Peng, W.-C., editors, *Advances in Knowledge Discovery and Data Mining*, volume 13935, pages 95–107. Springer Nature Switzerland, Cham.

[Diligenti et al., 2017] Diligenti, M., Gori, M., and Saccà, C. (2017). Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165.

[Doshi-Velez and Kim, 2017] Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv:1702.08608 [cs, stat]*.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences.

[Egho et al., 2015] Egho, E., Gay, D., Boulle, M., Voisine, N., and Clerot, F. (2015). A Parameter-Free Approach for Mining Robust Sequential Classification Rules. In *2015 IEEE International Conference on Data Mining*, pages 745–750, Atlantic City, NJ. IEEE.

[Elhage et al., 2022] Elhage, Nelson, Hume, T., Olsson, C., Nanda, N., Henighan, T., Johnston, S., ElShowk, S., Joseph, N., DasSarma, N., Mann, B., Hernandez, D., Askell, A., Ndousse, K., Jones, and Drain, D., Chen, A., Bai, Y., Ganguli, D., Lovitt, L., Hatfield-Dodds, Z., Kernion, J., Conerly, T., Kravec, S., Fort, S., Kadavath, S., Jacobson, J., Tran-Johnson, E., Kaplan, J., Clark, J., Brown, T., McCandlish, S., Amodei, D., and Olah, C. (2022). Softmax linear units. *Transformer Circuits Thread*.

[Evans and Grefenstette, 2018] Evans, R. and Grefenstette, E. (2018). Learning Explanatory Rules from Noisy Data.

[Forgy, 1982] Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.

[Fürnkranz et al., 2012] Fürnkranz, J., Gamberger, D., and Lavrač, N. (2012). *Foundations of Rule Learning*. Springer Science & Business Media.

[Fürnkranz and Kliegr, 2015] Fürnkranz, J. and Kliegr, T. (2015). A Brief Overview of Rule Learning. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., and Roman, D., editors, *Rule Technologies: Foundations, Tools, and Applications*, volume 9202, pages 54–69. Springer International Publishing, Cham.

[Fürnkranz and Widmer, 1996] Fürnkranz, J. and Widmer, G. (1996). Incremental Reduced Error Pruning.

[Geiger and Team, 2020] Geiger, L. and Team, P. (2020). Larq: An open-source library for training binarized neural networks. *Journal of Open Source Software*, 5(45):1746.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[Grisoni et al., 2019] Grisoni, F., Neuhaus, C. S., Hishinuma, M., Gabernet, G., Hiss, J. A., Kotera, M., and Schneider, G. (2019). De novo design

of anticancer peptides by ensemble artificial neural networks. *Journal of Molecular Modeling*, 25(5):112.

[Guyon and Elisseeff, 2003] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(null):1157–1182.

[Hailesilassie, 2016] Hailesilassie, T. (2016). Rule Extraction Algorithm for Deep Neural Networks: A Review.

[Helwegen et al., 2019] Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.-T., and Nusselder, R. (2019). Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[Hemker et al., 2023] Hemker, K., Shams, Z., and Jamnik, M. (2023). CGXplain: Rule-Based Deep Neural Network Explanations Using Dual Linear Programs.

[Hitzler et al., 2022] Hitzler, P., Eberhart, A., Ebrahimi, M., Sarker, M. K., and Zhou, L. (2022). Neuro-symbolic approaches in artificial intelligence. *National Science Review*, 9(6):nwac035.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, 9:1735–80.

[Hoefler et al., 2021] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. (2021). Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124.

[Hühn and Hüllermeier, 2009] Hühn, J. and Hüllermeier, E. (2009). FURIA: An algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319.

[Kahneman, 2011] Kahneman, D. (2011). *Thinking, Fast and Slow*. macmillan.

[Karimi and Hamilton, 2010] Karimi, K. and Hamilton, H. J. (2010). Generation and Interpretation of Temporal Decision Rules.

[Kaul et al., 2017] Kaul, A., Maheshwary, S., and Pudi, V. (2017). AutoLearn — Automated Feature Generation and Selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226.

[Kautz, 2022] Kautz, H. A. (2022). The third AI summer: AAAI Robert S. Engelmore Memorial Lecture. *AI Magazine*, 43(1):105–125.

[Keijzer and Babovic, 1999] Keijzer, M. and Babovic, V. (1999). Dimensionally Aware Genetic Programming. *Gecco-99: Proceedings of the Genetic and Evolutionary Computation Conference*.

[Khalitov et al., 2023] Khalitov, R., Yu, T., Cheng, L., and Yang, Z. (2023). ChordMixer: A Scalable Neural Attention Model for Sequences with Different Length. In *The Eleventh International Conference on Learning Representations*.

[Kusters et al., 2022] Kusters, R., Kim, Y., Collery, M., Marie, C. d. S., and Gupta, S. (2022). Differentiable Rule Induction with Learned Relational Features. *arXiv:2201.06515 [cs, stat]*.

[Lamb et al., 2021] Lamb, L. C., Garcez, A., Gori, M., Prates, M., Avelar, P., and Vardi, M. (2021). Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective.

[Lample and Charton, 2019] Lample, G. and Charton, F. (2019). Deep Learning for Symbolic Mathematics. *arXiv:1912.01412 [cs]*.

[Li and Zaiane, 2017] Li, J. and Zaiane, O. R. (2017). Exploiting statistically significant dependent rules for associative classification. *Intell. Data Anal.*

[Li et al., 2021] Li, X., Xiong, H., Li, X., Wu, X., Zhang, X., Liu, J., Bian, J., and Dou, D. (2021). Interpretable Deep Learning: Interpretation, Interpretability, Trustworthiness, and Beyond. *arXiv:2103.10689 [cs]*.

[Lin et al., 2020] Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. (2020). Dynamic model pruning with feedback. In *International Conference on Learning Representations*.

[Liu et al., 1998] Liu, B., Hsu, W., and Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 80–86, New York, NY. AAAI Press.

[Louizos et al., 2018] Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*.

[Manhaeve et al., 2018] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). DeepProbLog: Neural Probabilistic Logic Programming. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Marcus and Davis, 2019] Marcus, G. and Davis, E. (2019). *Rebooting AI: Building Artificial Intelligence We Can Trust*. Pantheon Books, USA.

[Nwegbu et al., 2022] Nwegbu, N., Tirunagari, S., and Windridge, D. (2022). A novel kernel based approach to arbitrary length symbolic data with application to type 2 diabetes risk. *Scientific Reports*, 12:4985.

[Otero et al., 2003] Otero, F. E. B., Silva, M. M. S., Freitas, A. A., and Nievola, J. C. (2003). Genetic Programming for Attribute Construction in Data Mining. In Goos, G., Hartmanis, J., van Leeuwen, J., Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *Genetic Programming*, volume 2610, pages 384–393. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Persia and Guimarães, 2023] Persia, C. and Guimarães, R. (2023). RIDDLE: Rule Induction with Deep Learning. *Proceedings of the Northern Lights Deep Learning Workshop*, 4.

[Petersen et al., 2022] Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. (2022). Deep Differentiable Logic Gate Networks.

[Prechelt, 1997] Prechelt, L. (1997). Connection pruning with static and adaptive pruning schedules. *Neurocomputing*, 16(1):49–61.

[Qiao et al., 2021] Qiao, L., Wang, W., and Lin, B. (2021). Learning accurate and interpretable decision rule sets from neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4303–4311.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Quinlan, 1996] Quinlan, J. R. (1996). Improved Use of Continuous Attributes in C4.5.

[Ratle and Sebag, 2001] Ratle, A. and Sebag, M. (2001). Grammar-guided genetic programming and dimensional consistency: Application to nonparametric identification in mechanics. *Applied Soft Computing*, 1(1):105–118.

[Riegel et al., 2020] Riegel, R., Gray, A., Luus, F., Khan, N., Makondo, N., Akhalwaya, I. Y., Qian, H., Fagin, R., Barahona, F., Sharma, U., Ikbal, S., Karanam, H., Neelam, S., Likhyani, A., and Srivastava, S. (2020). Logical Neural Networks.

[Rocktäschel and Riedel, 2017] Rocktäschel, T. and Riedel, S. (2017). End-to-End Differentiable Proving.

[Rudin, 2019] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

[Sarker et al., 2021] Sarker, M. K., Zhou, L., Eberhart, A., and Hitzler, P. (2021). Neuro-Symbolic Artificial Intelligence: Current Trends.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Sondhi, 2009] Sondhi, P. (2009). Feature construction methods : A survey.

[Sood et al., 2020] Sood, N., Bindra, L., and Zaiane, O. (2020). Bi-Level Associative Classifier Using Automatic Learning on Rules. In Hartmann, S., Küng, J., Kotsis, G., Tjoa, A. M., and Khalil, I., editors, *Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 201–216, Cham. Springer International Publishing.

[Srinivas et al., 2017] Srinivas, S., Subramanya, A., and Babu, R. V. (2017). Training Sparse Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 455–462, Honolulu, HI, USA. IEEE.

[Stańczyk et al., 2020] Stańczyk, U., Zielosko, B., and Baron, G. (2020). Discretisation of conditions in decision rules induced for continuous data. *PLoS ONE*, 15(4):e0231788.

[Sushil et al., 2018] Sushil, M., Šuster, S., and Daelemans, W. (2018). Rule induction for global explanation of trained models. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 82–97, Brussels, Belgium. Association for Computational Linguistics.

[Thompson et al., 2022] Thompson, N. C., Greenewald, K., Lee, K., and Manso, G. F. (2022). The Computational Limits of Deep Learning.

[Udrescu and Tegmark, 2020] Udrescu, S.-M. and Tegmark, M. (2020). AI Feynman: A Physics-Inspired Method for Symbolic Regression. *arXiv:1905.11481 [hep-th, physics:physics]*.

[van Krieken et al., 2022] van Krieken, E., Acar, E., and van Harmelen, F. (2022). Analyzing Differentiable Fuzzy Logic Operators. *Artificial Intelligence*, 302:103602.

[Wang et al., 2022] Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2022). Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 small.

[Willard et al., 2022] Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V. (2022). Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems.

[Wojtusiak, 2012] Wojtusiak, J. (2012). Rule Learning. In Seel, N. M., editor, *Encyclopedia of the Sciences of Learning*, pages 2909–2911. Springer US, Boston, MA.

[Xing et al., 2010] Xing, Z., Pei, J., and Keogh, E. (2010). A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48.

[Yang et al., 2023] Yang, C., Wang, L., Gao, K., and Li, S. (2023). Reinforcement Logic Rule Learning for Temporal Point Processes.

[Yin and Han, 2003] Yin, X. and Han, J. (2003). CPAR: Classification based on Predictive Association Rules. In *SDM*.

[Yu et al., 2023] Yu, D., Yang, B., Liu, D., Wang, H., and Pan, S. (2023). A Survey on Neural-symbolic Learning Systems.

[Zhou et al., 2013] Zhou, C., Cule, B., and Goethals, B. (2013). Itemset Based Sequence Classification. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Salinesi, C., Norrie, M. C., and Pastor, Ó., editors, *Advanced Information Systems Engineering*, volume 7908, pages 353–368. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Zhou et al., 2015] Zhou, C., Cule, B., and Goethals, B. (2015). Pattern Based Sequence Classification. *IEEE Transactions on Knowledge and Data Engineering*, 28:1–1.

[Zhu and Gupta, 2017] Zhu, M. and Gupta, S. (2017). To prune, or not to prune: Exploring the efficacy of pruning for model compression.

[Zilke et al., 2016] Zilke, J. R., Loza Mencía, E., and Janssen, F. (2016). DeepRED – Rule Extraction from Deep Neural Networks. In Calders, T., Ceci, M., and Malerba, D., editors, *Discovery Science*, Lecture Notes in Computer Science, pages 457–473, Cham. Springer International Publishing.

ECOLE
DOCTORALE

Titre : Apprentissage de règles de classification expressives notamment à partir de données séquentielles.

Mots clés : apprentissage automatique, IA interpretable, IA explicative, neuro-symbolique, apprentissage de règles, données séquentielles

Résumé : Au cours des dernières décennies, l'apprentissage automatique, et en particulier avec les réseaux de neurones, a fait d'énormes progrès pour résoudre des problèmes de classification dans différents domaines tels que la santé, la détection des fraudes ou la reconnaissance d'images. Ces modèles sont capables d'apprendre à partir de différents types de données, allant des images aux séries temporelles, et d'atteindre une précision de classification impressionnante. Cependant, leurs décisions sont difficiles, voire impossibles à comprendre par un être humain. Les méthodes basées sur des règles, quant à elles, sont interprétables, lisibles par l'homme et ont été largement adoptées dans différents domaines industriels avec les Business Rule Management Systems (BRMS) ou systèmes de gestion des règles métier. En pratique, cependant, ces règles sont écrites manuellement par des experts. L'une des raisons pour laquelle les règles écrites manuellement ne peuvent pas être facilement remplacées par des modèles de règles apprises à partir de données, est que les modèles d'apprentissage de règles ne sont pas capables d'apprendre des règles aussi expressives, avec des concepts de haut niveau et une grammaire complexe. De plus, en raison d'un manque de représentations latentes, les méthodes d'apprentissage basées sur des règles sont moins performantes que les réseaux neuronaux de l'état de l'art.
Dans cette thèse, nous proposons une approche de bout en bout basée sur un réseau de neurones permettant d'apprendre des règles expressives pour des problèmes de classification. Différents niveaux d'expressivité des règles sont présentés et évalués sur de nouvelles données synthétiques et sur certains ensembles de données existants. Tout d'abord, l'apprentissage d'expressions sous la forme normale disjonctive avec un réseau neuronal (modèle de base) est étudié. Ensuite, des extensions pour prendre en charge les données séquentielles sont introduites avec une approche récursive et une approche convolutive. Enfin, le modèle est étendu pour apprendre des règles plus expressives avec des fonctions d'agrégation prédéfinies et des règles de grammaire complexes.

Title : Expressive classification rule learning with an emphasis on learning from sequential data.

Keywords : machine learning, interpretable AI, explanatory AI, neuro-symbolic, rule learning, sequential data

Abstract : During the last decades, machine learning and in particular neural networks have made tremendous progress on classification tasks for a variety of fields such as healthcare, fraud detection or image recognition. They are able to learn from various data types ranging from images to time series and achieve impressive classification accuracy. However, their decisions are difficult or impossible to understand by a human. Rule-based methods on the other end, are interpretable, human-readable and have been widely adopted in different industrial fields with Business Rule Management Systems (BRMS). In practice however, those rules are manually written by experts. One of the reasons manually-written rule models cannot easily be replaced with learned rule models is that rule-based learning models are not able to learn as expressive rules with higher-level concepts and complex grammar. Moreover, due to the lack of latent representations, rule-based learning methods underperform w.r.t. state-of-the-art neural networks.
In this thesis, we propose an end-to-end neural-based approach to learn expressive rules for classification problems. Different levels of expressiveness in rules are presented, implemented and evaluated on some existing datasets and new synthetic ones proposed as new benchmarks for binary classification rule learning. First, the learning of basic disjunctive normal form with a neural network (base model) is studied. Second, extensions to support sequential data are introduced with a recursive and a convolutional approaches. Finally, the model is extended to learn more expressive rules with predefined aggregation functions and overall complex grammar rules.

Institut Polytechnique de Paris
91120 Palaiseau, France