A Neuro-symbolic Approach to Argument Comparison in Structured Argumentation

Damián Ariel Furman[1], Stephanie Anneris Malvicini[2], Maria Vanina Martinez[3], Paulo Shakarian[4], Gerardo Ignacio Simari[2,4,*] and Yamil Osvaldo Soto[2]

[1]*Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Intendente Güiraldes 2160, Ciudad Universitaria, Buenos Aires, Argentina.*

[2]*Departamento de Ciencias e Ingenieria de la Computacion, Universidad Nacional del Sur (UNS) & Instituto de Ciencias e Ingenieria de la Computacion (ICIC UNS–CONICET), San Andres 800, Bahia Blanca, Buenos Aires, Argentina.*

[3]*Artificial Intelligence Research Institute (IIIA-CSIC), Carrer de Can Planas, Barcelona, Catalonia, Spain.*

[4]*Arizona State University, Tempe, AZ, United States of America*

Abstract

Defeasible Logic Programming (DeLP) is a structured argumentation formalism that uses a dialectical process to decide between contradictory conclusions. Such conclusions are supported by arguments, which are compared using a comparison criterion, to decide which one prevails in conflict situations. The definition of a formal comparison criterion is a central problem in structured argumentation, which is typically assumed to be provided by the user or knowledge engineer. In this work, we propose an integration between an argumentative approach to defeasible reasoning, such as DeLP, and machine learning models. Concretely, our goal is to train a neural network to learn a comparison criterion between arguments given a training set comprised of pairs of arguments labeled with which one prevails. We conducted several experiments, using a synthetic DeLP program generator, in order to assess the performance of a neural architecture under different kinds of DeLP programs. Our results show that under specific circumstances, a comparison criterion for arguments can be successfully learned by data-driven models.

Keywords

Structured Argumentation, Defeasible Reasoning, Defeasible Logic Programming, Neuro-symbolic reasoning, Neuro-symbolic learning,

1. Introduction

The integration between machine learning and argumentation has received some attention in the past; though interesting results have been obtained, there are still many aspects to investigate; for a survey of works on this subject, see [1]. In this work, we focus on combining

7th Workshop on Advances in Argumentation in Artificial Intelligence (AI^3), November 06–09, 2023, Rome, Italy

*Corresponding author.

dfurman@dc.uba.ar (D. A. Furman); StephanieMalvicini@gmail.com (S. A. Malvicini); vmartinez@iiia.csic.es (M. V. Martinez); pshak02@asu.edu (P. Shakarian); gis@cs.uns.edu.ar (G. I. Simari); yamil.soto@cs.uns.edu.ar (Y. O. Soto)

https://labs.engineering.asu.edu/labv2/about-paulo-shakarian/ (P. Shakarian); http://www.cs.uns.edu.ar/~gis/ (G. I. Simari)

0000-0003-4570-1364 (D. A. Furman); 0000-0002-9421-8566 (M. V. Martinez); 0000-0002-3159-4660 (P. Shakarian); 0000-0003-3185-4992 (G. I. Simari); 0000-0001-8695-9949 (Y. O. Soto)

neural networks and an argumentative approach to defeasible reasoning. To explain why this integration is of our interest, we are going to briefly discuss some of the main characteristics of both parts.

Machine learning [2] is an exciting field that has matured into being capable of delivering important applications in the recent years after decades of slower progress. Today, much of the work in areas such as visual processing, language, and speech recognition is at least in part reliant on machine learning. In particular, neural networks [3], as a particular machine learning model, are capable of intrinsic massively parallel processing, which makes them suitable for several complex domains.

On the Knowledge Representation and Reasoning side, defeasible reasoning [4] is, intuitively, the process of *tentatively* inferring or deriving logical conclusions from available information—*i.e.*, since the connection between premises and conclusions is a tentative one, then we can evaluate the rejection of our conclusions in the presence of information that contradicts them. In this work, we adopt an argumentative approach to defeasible reasoning, where queries are analyzed in a *dialectical process* that exhaustively considers arguments for and against specific answers in search of *warrants*, which refers to deciding how available information supports specific conclusions.

The combination of neural networks and argumentation is attractive because both fields can benefit from the capabilities of the other. On the one hand, neural networks can improve the performance of argumentative systems by allowing them to take advantage of available information (*e.g.*, in real-world datasets) and to adapt to dynamic environments, with high volumes of data and where answers must be given quickly. On the other hand, argumentation can help neural networks deal better with inconsistent and incoherent information, can increase their explanatory power, and can also aid them in expressing specific domain knowledge.

In this work, our goal is to train a neural network to compare arguments and decide which one prevails in a conflict. For that purpose, we need the network to learn a specific component called a *comparison criterion*, which compares two arguments and decides which one prevails. This can benefit argumentative systems by allowing them to learn a new comparison criterion from examples, in a dynamic way and relevant to the particular situations that different programs aim to deal with. In general, it is not always simple to define a comparison criterion that fits the complexity of arguments in a given domain, so this task is often assumed to be carried out by a knowledge engineer. An approach like ours can therefore improve the performance of argumentative systems, for example in legal reasoning, where they could learn from datasets related to jurisprudence. While prior work on the integration of neural and symbolic AI architectures has introduced parameterized constants [5] and parameterized operators [6], we believe this is the first time a comparison criterion has been parameterized, thus advancing the efforts in the field.

Our two main hypotheses for this preliminary work are:

1. A neural network can learn a comparison criterion and decide, given two arguments, which one is preferred (if any), and
2. the complexity of the comparison criterion and the generated programs, and the number of examples affect the performance of the network.

This work is organized as follows. In Section 2 we review some basic concepts about the

argumentation formalism that we use, called *Defeasible Logic Programming* (DeLP). In Section 3 we describe several aspects about the approach taken to arrive at an adequate neural encoding of arguments. Results obtained from the experimentation are shown in Section 4. In Section 5 we discuss related work, and finally, we summarize our main results and describe future lines of research in Section 6.

2. Background

We now recall some basic concepts on Defeasible Logic Programming, which from now on we refer to as DeLP for short; for the full details of the formalism, we refer the reader to [7].

2.1. Defeasible Logic Programming (DeLP)

DeLP [7] is a formalism that combines logic programming and defeasible argumentation. A DeLP program $P = (\Pi, \Lambda)$ is comprised of a set of facts and strict rules (Π), and a set of defeasible rules (Λ). In this work, we consider an extension of DeLP [8], called PreDeLP, that incorporates *presumptions*, which can be thought of as a kind of defeasible facts.

In the following, we briefly recall the most relevant concepts for PreDeLP from [8]. In the PreDeLP language, a literal L is a ground atom or a negated ground atom. We represent strong negation with "$\sim$" and say that L and $\sim L$ are complementary. A *strict rule* R (resp., *defeasible rule*) has the form:

$$L \leftarrow L_1, \ldots, L_n \text{ (resp., } L \prec L_1, \ldots, L_n),$$

where L is the $head$ of R, and $L_1, \ldots, L_n$, with $n \geqslant 1$, is the $body$ of R. *Facts* are ground literals representing atomic information. Strict rules represent a strong relation between $body$ and $head$. Both facts and strict rules constitute strict (sound) information.

A defeasible rule $head \prec body$ represents a weaker connection between $body$ and $head$. It can be understood as expressing that "reasons to believe in the antecedent $body$ provide reasons to believe in the consequent $head$" [9], but it may be the case that $body$ is true and $head$ is not. Note that the symbols $\leftarrow$ and $\prec$ denote meta-relations between a literal and a set of literals, and have no interaction with language symbols. As in Logic Programming, strict and defeasible rules are not conditionals nor implications, they are inference rules [7, 10].

A *fact* (resp., *presumption*) is a strict (resp., defeasible) rule with an empty body, denoted with L (resp., $L \prec$). Intuitively, presumptions are defined as pieces of information that are tentatively taken to be true, usually in the absence of acceptable reasons to the contrary. Since they express reasons to believe in some information, they represent weaker assertions than facts.

Definition 1 (PreDeLP)**.** *A* PreDeLP program P*, denoted with* $(\Omega, \Theta, \Delta, \Phi)$*, is a set of strict rules* Ω*, facts* Θ*, defeasible rules* Δ*, and presumptions* Φ*.*

Next, we introduce the concept of annotated derivation, or just derivation, where we adopt the definition given in [8].

Definition 2. *Let* P *be a PreDeLP program and* L *a literal. An* annotated derivation ∂ *of* L *from* P*, consists of a finite sequence of rules, facts, and possibly presumptions* $[R_1, \ldots, R_n]$*, where* L *is*

1. *a fact* R_n,
2. *a presumption* R_n, *or*
3. *the head of the rule* R_n. *Furthermore, if a rule* R_i *is in the sequence, then its body* $B_1, \ldots, B_k$, *is such that for all* B_j, *with* $1 \leqslant j \leqslant k$, B_j *is a fact, a presumption or it appears as the head* L_m, *for some rule* R_m *with* $1 \leqslant m < i$.

A derivation ∂ is *strict* when neither presumptions nor defeasible rules occur in ∂; otherwise, ∂ is *defeasible*. A literal L is *strictly derived* from P, denoted $P \vdash L$, if there exists a strict derivation for L from P; on the other hand, L is *defeasibly derived* from P, denoted $P \mathrel{|\!\sim} L$, if there exists a defeasible derivation for L from P and no strict derivation exists. A derivation ∂ for L is *minimal* if no proper sub-derivation ∂' of ∂ (*i.e.*, ∂ subsumes ∂') is also a derivation of L. Considering minimal derivations avoids the insertion of unnecessary elements that will weaken its ability to support the conclusion by possibly introducing needless points of conflict. Given a derivation ∂ for L, there exists at least one minimal sub-derivation ∂' for L.

A PreDeLP program $P = (\Omega, \Theta, \Delta, \Phi)$ is contradictory if there exist derivations for two complementary literals. We denote with $\Pi = (\Omega, \Theta)$ the *strict part* of P, and assume that the sub-program Π is *non-contradictory*. Two literals L_1 and L_2 *disagree* w.r.t. P if $\Pi \cup \{L_1, L_2\}$ is contradictory.

Queries and Arguments. A query is issued to a program $P = (\Omega, \Theta, \Delta, \Phi)$ in the form of a ground literal α. The dialectical process used in deciding if α is *warranted* involves the construction and evaluation of arguments that either support or interfere with the query.

Definition 3. *Given a program* $P = (\Omega, \Theta, \Delta, \Phi)$, *let* α *be a ground literal, and* ∂ *a derivation for* α *obtained from* P. *An argument for* α constructed from P, *denoted with* $\langle \mathcal{A}, \alpha \rangle$, *is the set of facts, presumptions, and rules (strict and defeasible) used in the derivation* ∂.

Thus, we have that an argument $\langle \mathcal{A}, \alpha \rangle$ is a minimal non-contradictory set of facts, presumptions, and rules, strict and defeasible, contained in P.

Given a program P, a literal α is *warranted* if and only if there exists a non-defeated argument $\langle \mathcal{A}, \alpha \rangle$ supporting α. In order to establish whether $\langle \mathcal{A}, \alpha \rangle$ is a non-defeated argument, *defeaters* for $\langle \mathcal{A}, \alpha \rangle$ are considered, *i.e.*, counterarguments that by some criterion are preferred to $\langle \mathcal{A}, \alpha \rangle$. An argument $\langle \mathcal{A}, \alpha \rangle$ is a *counterargument* for $\langle \mathcal{B}, \beta \rangle$ if and only if $\langle \mathcal{A}, \alpha \rangle \cup \langle \mathcal{B}, \beta \rangle \cup \Pi$ is contradictory. Given a *preference criterion* $\succ$, an argument $\langle \mathcal{A}, \alpha \rangle$ is called a *proper defeater* of an argument $\langle \mathcal{B}, \beta \rangle$ if it is preferred to $\langle \mathcal{B}, \beta \rangle$ according to $\succ$, or is called a *blocking defeater* of $\langle \mathcal{B}, \beta \rangle$ if it is equally preferred or is incomparable with $\langle \mathcal{B}, \beta \rangle$ according to $\succ$. In both cases, $\langle \mathcal{A}, \alpha \rangle$ *defeat* $\langle \mathcal{B}, \beta \rangle$. Figure 1 shows an example where an argument $\langle \mathcal{A}, \delta \rangle$ is, according to an unspecified comparison criterion $\succ$, a proper defeater of argument $\langle \mathcal{B}, \sim\delta \rangle$.

The dialectical analysis that DeLP carries out to decide if an atom is warranted is outside the scope of this paper; we discuss the main points, and refer the reader to [7] for more details. Since there may be several defeaters for an argument, many acceptable argumentation lines could arise from one argument, leading to a tree structure. An *argumentation line* $\partial = [\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n]$ is a sequence of arguments in which every element of the sequence is a defeater of its predecessor (except for the first argument). A *dialectical tree* provides a structure for considering all possible acceptable argumentation lines that can be generated for deciding whether an argument is defeated (each path from the root to a leaf corresponds to a different acceptable argumentation

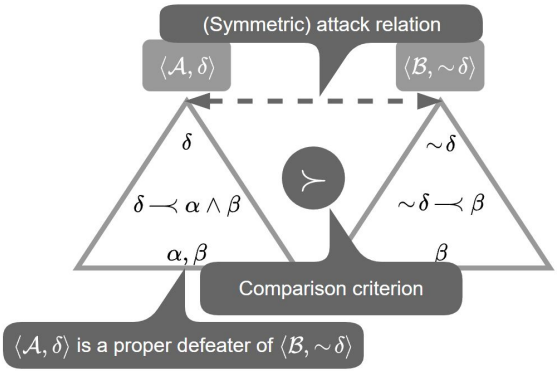

Figure 1: Argument $\langle \mathcal{A}, \delta \rangle$ is a proper defeater of argument $\langle \mathcal{B}, \sim\delta \rangle$ according to criterion $\succ$.

line). This is called a dialectical tree because it represents an exhaustive dialectical analysis for the argument at its root.

Given a literal α and an argument $\langle \mathcal{A}, \alpha \rangle$ from a program P, to decide whether α is warranted, every node in the tree $\mathcal{T}(\langle \mathcal{A}, \alpha \rangle)$ is recursively marked as D (defeated) or U (undefeated), obtaining a marked dialectical tree $\mathcal{T}'(\langle \mathcal{A}, \alpha \rangle)$ where:

1. All leaves in $\mathcal{T}'(\langle \mathcal{A}, \alpha \rangle)$ are marked as U; and
2. let N be an inner node of $\mathcal{T}'(\langle \mathcal{A}, \alpha \rangle)$; then, N is marked as U if and only if every child of N is marked as D, *i.e.*, all defeaters of N are defeated.

Thus, node N is marked as D if and only if it has at least one child marked as U, *i.e.*, at least one defeater of N is not defeated. Given an argument $\langle \mathcal{A}, \alpha \rangle$ obtained from P, if the root of $\mathcal{T}'(\langle \mathcal{A}, \alpha \rangle)$ is marked as U, then $\mathcal{T}'(\langle \mathcal{A}, \alpha \rangle)$ *warrants* α, and α is *warranted* from P.

Given a DeLP program P, a query α returns one of the following four possible answers: *YES* if α is warranted from P, *NO* if the complement of α is warranted from P, *UNDECIDED* if neither α nor its complement are warranted from P, or *UNKNOWN* if α is not in the language of the program P.

2.2. Comparison Criteria

As mentioned above, when two arguments attack each other, we must analyze which one prevails. Given an argument $\langle \mathcal{A}, \alpha \rangle$ and a counter-argument $\langle \mathcal{B}, \beta \rangle$, a comparison criterion is used to determine if $\langle \mathcal{A}, \alpha \rangle$ is preferred to $\langle \mathcal{B}, \beta \rangle$ and, therefore, defeats $\langle \mathcal{B}, \beta \rangle$. The definition of such a formal criterion is a central problem in structured argumentation, *i.e.*, in those systems where the defeat relation must be computed from the structure of arguments. Although the comparison criterion used in DeLP is modular, the default one is *Generalized Specificity* [9, 11]. This criterion, intuitively, favors arguments with greater information content (classic specificity) or with less use of rules (a more direct derivation)—in other words, an argument is deemed better than another if it is more precise or more concise.

In the presence of *presumptions*, Generalized Specificity does not always have the intended results; for that reason, other preference criteria have been developed, such as *Presumption-enabled Specificity* [8]. According to this criterion, the accumulation of presumptive information

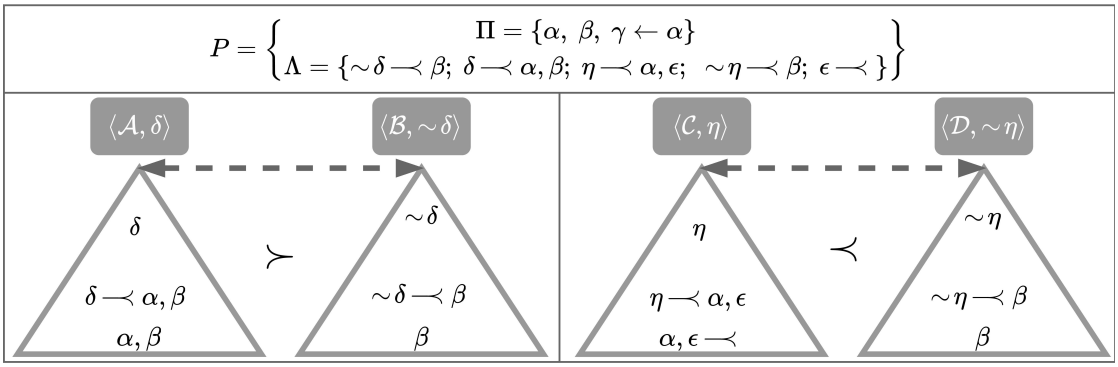

Figure 2: An example of a DeLP program P from which we can build the arguments shown and the results of the comparison between those arguments according to the selected criterion. Generalized Specificity is used on the left, and Presumption-enabled Specificity on the right.

in an argument weakens both the argument itself and its conclusion, meaning that given two presumptive arguments, if one uses a subset of the presumptions used by the other one, the former is considered to be a *better argument*. Figure 2 shows a simple example of the use of each criterion.

Towards other types of comparison criteria. The criteria discussed above are examples of domain-independent argument comparison criteria; however, in real life applications it may be necessary to introduce a criterion that depends on the particular domain and usage of information. In [7], an alternative general criterion is proposed assuming it is possible to elicit a preference order among the set of defeasible rules in the program, which is later lifted to a preference order among arguments that can be built from it.

In [12], an approach to handle multiple argument preference criteria is proposed in the setting of argumentation-based recommender systems using DeLP as the underlying knowledge representation and reasoning framework. The proposal allows to change the criterion that can be used in the comparison process, and even to use several criteria simultaneously that can be combined through adequate operators. Nevertheless, establishing which criteria should be used, and determining whether it needs to change over time, is not a trivial task. In the next section, we develop our proposal that seeks to learn domain (or application) specific criteria using neural networks.

3. Training a Neural Network to Compare Arguments

3.1. General Overview

Our methodology for training neural network models is comprised of three steps. First, we use a synthetic DeLP program generator [13] to produce a set of programs. Then, for each generated program, and using a DeLP solver [10, 14], we generate a list of pairs of arguments compared during the inference process indicating which one was chosen by the solver as a defeater. Finally, we train the neural network, having as input the pairs of arguments and a scalar as output, indicating which argument is preferred (or if neither is, which occurs in the

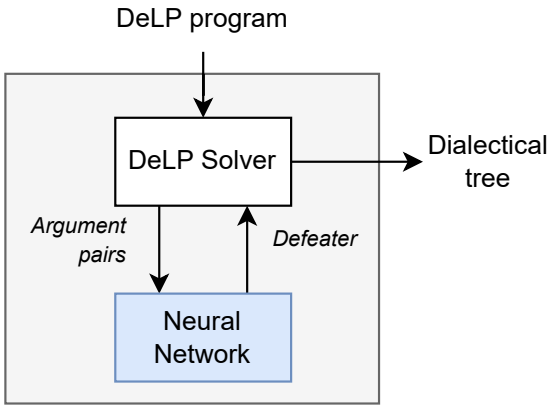

Figure 3: A high-level view of how a neural network can be integrated with the solver in the dialectical process that lies at the heart of argumentation-based query answering in DeLP.

case of blocking defeaters).

Figure 3 illustrates our vision of how a neural network and the DeLP solver can be integrated, where the former is trained to learn a criterion and carry out the comparison between arguments, while the latter carries out the dialectical analysis.

3.2. Synthetic Program Generation and Experimental Settings

For the programs obtained from the DeLP generator, we consider two criteria: *Generalized Specificity* and *Presumption-enabled Specificity*. In addition, we explored two types of program settings: *complexity* (simple or complex programs) and the *presence of blocking defeaters*. The complexity of programs depends mainly on the following parameters:

- *MAX_BODYSIZE* (sets the highest value in the average length of the defeasible derivation of any argument),
- *MIN_ARGSLEVEL* (sets the number of dialectical trees),
- *RAMIFICATION* (sets the number of argumentation lines), and
- *TREE_HEIGHT* (sets the length of argumentation lines).

We suggest the reader to see [13] for more details on the synthetic DeLP program generating processes. The presence of blocking defeaters indicates whether or not they are considered when generating the pairs of arguments to be compared.

For each setting combination, three datasets were generated with different sizes: 200, 500, and 1000 programs. Table 1 shows the average number of argument pairs per program based on the program configuration described above.

3.3. Encoding Arguments

In order to be amenable to neural training methods, we need to develop a vector encoding of arguments obtained from DeLP programs, where the encoding is done in such a way that the connection between internal components is adequately represented.

Complexity	Blocking	#Argument pairs per program
Simple	×	9.63
Simple	✓	18.80
Complex	×	57.07
Complex	✓	243.52

Table 1
Average number of pairs of arguments per program in the synthetically generated dataset.

The network input is constructed by encoding a pair of conflicting arguments into a single tensor, with the following conventions (illustrated in Figure 4):

- Value "1" is used to represent defeasible rules, while "-1" is used to represent strict rules.
- Each *atom* is represented with an *id* consisting of an integer value strictly greater than 2. The same integer will be used to represent the same atom in the context of a pair of conflicting arguments. The *negation* of an atom is represented by the negative counterpart of the number representing that atom.
- Rules and atoms are separated using "2", while arguments are separated using value"-2".
- Arguments have a fixed length so if the representation of the argument is smaller than that fixed length, zeros are used to fill the unused space and the neural network ignores them.
- For the output, we have three possible values:
  - 0 if the first argument is preferred over the second one,
  - 1 if the second argument is preferred over the first one, and
  - 2 if no argument is preferred over the other, *i.e.*, the arguments are blocking defeaters.

As mentioned, we developed this encoding because it maintains the relationship between the inner components of the arguments, and it allows each argument to be represented as a tensor of the same length. Algorithms 1 and 2 describe the full processes of encoding a single argument and the construction of the input, respectively.

The following example shows how a pair of conflicting arguments are encoded and how the input is built from both encodings. Figure 4 provides a graphical illustration of the example.

Example 1. *Let $\langle \mathcal{A}, a_8 \rangle$ and $\langle \mathcal{B}, \sim a_8 \rangle$ be two conflicting arguments.*

In this case, literal a_8 is represented with value 7, a_0 with 8, and d_9 with 9, while a_8 is represented with -7. The symbol "$\prec$" is represented with value 1, while -1 is used after the value 8 to indicate that a_0 is a fact. Rules are separated from facts and presumptions using the value 2, and both arguments are concatenated on the final input vector adding a value of -2 in between.

$$
\begin{array}{ll}
\langle \mathcal{A}, a_8 \rangle & : [(a_8 \prec a_0), a_0] \\
\textit{Encoding of } \langle \mathcal{A}, a_8 \rangle & : [7, 1, 8, 2, 8, -1, 0, ..., 0] \\
\langle \mathcal{B}, \sim a_8 \rangle & : [(\sim a_8 \prec d_9), (d_9 \prec true)] \\
\textit{Encoding of } \langle \mathcal{B}, \sim a_8 \rangle & : [-7, 1, 9, 2, 9, 1, 0, ..., 0]
\end{array}
$$

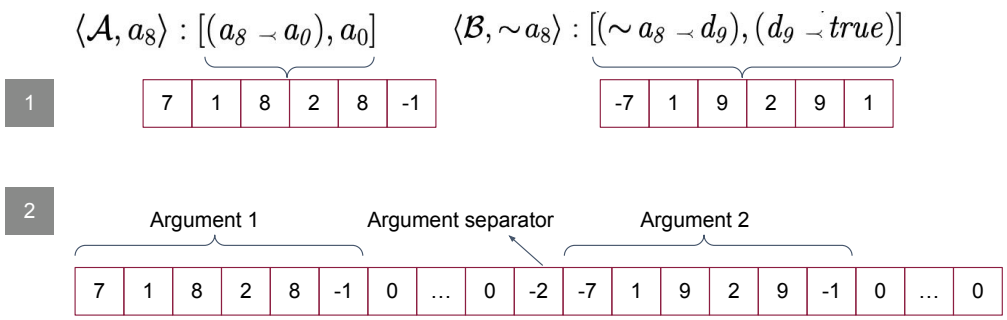

Figure 4: Encoding of arguments to be used as input to train neural networks. (1) shows the final individual encoding of arguments $\langle \mathcal{A}, \alpha_8 \rangle$ and $\langle \mathcal{B}, \sim \alpha_8 \rangle$. (2) shows the final neural network input encoding (generalized to multiple arguments).

Algorithm 1: EncodeArgument

Input: *Argument*

encodedArgument ← [ ]

for each *element* **in** *Argument* **do**

  encodedHead ← *LiteralToInteger*(*element.head*)

  encodedArgument.append(*encodedHead*)

  if *element.isFact* **or** *element.isStrictRule* **then**

    encodedRuleType ← −1

  else

    encodedRuleType ← 1

  encodedArgument.append(*encodedRuleType*)

  for each *literal* **in** *element.body* **do**

    encodedArgument.append(*LiteralToInteger*(*literal*))

  if *element* **is not** *Argument.last* **then**

    encodedArgument.append(2)

return *encodedArgument*

For the output, the value 0 *represents that* $\langle \mathcal{A}, a_8 \rangle$ *defeats* $\langle \mathcal{B}, \sim a_8 \rangle$*:*

$$\begin{aligned} &\textit{Input} \quad : [7, 1, 8, 2, 8, -1, 0, ..., 0, -2, -7, 1, 9, 2, 9, 1, 0, ..., 0] \\ &\textit{Output} \quad : 0 \end{aligned}$$

After presenting the neural encoding, in the next section we discuss the neural architecture proposed and used in our empirical evaluuation.

3.4. Neural Network Architecture

For this preliminary proposal, we used a simple neural network architecture with $3,000$ parameters (10 linear, fully connected hidden layers with 300 nodes each). The output of the neural

Algorithm 2: EncodeNetworkInput

Input: $Argument1, Argument2$

$encodedArgument1 \leftarrow EncodeArgument(Argument1)$

$encodedArgument2 \leftarrow EncodeArgument(Argument2)$

$input \leftarrow FillToMaxLength(encodedArgument1)$

$input.append(-2)$

$input.append(FillToMaxLength(encodedArgument2))$

return $input$

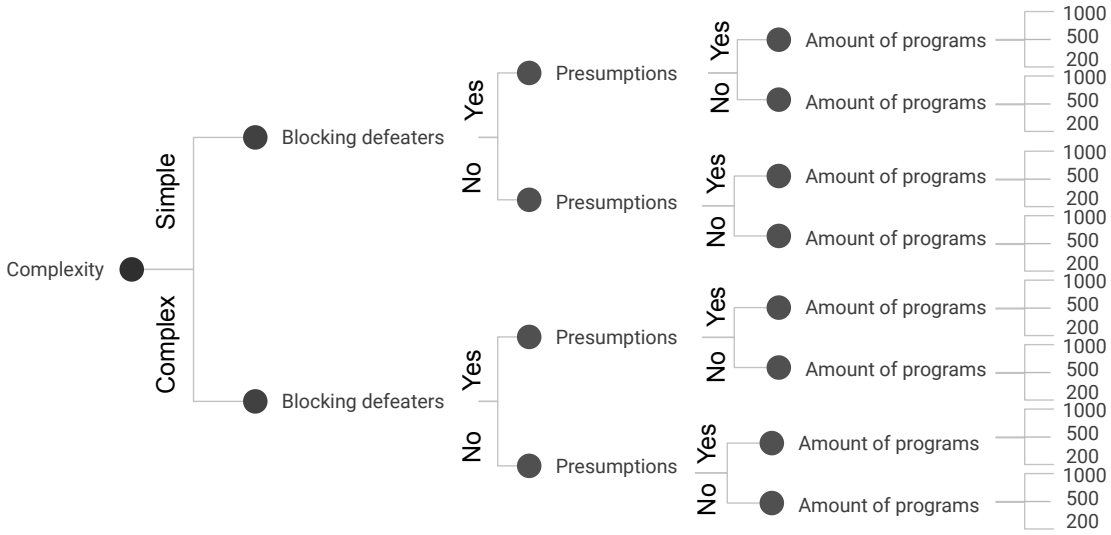

Figure 5: An overview of the different datasets used for experimental evaluation.

network depends on the characteristics of the arguments being processed:

- In the cases where only proper defeaters are present, the network uses a Sigmoid function to output a value between 0 and 1. For evaluation, the output value is rounded to the closest integer indicating if the defeating argument is the first or second one.
- In the cases where also blocking defeaters are present, the network outputs three values and uses a Softmax function to represent the *probability* assigned to each of the three possible classes: the first argument wins, the second wins, or they block each other.

In the next section, we discuss the details of our experimental evaluation using this setup.

4. Experimental Evaluation

Dataset. Different datasets were created combining all possible values of the following settings: with single or complex programs, with or without blocking defeaters, using Presumption-enabled

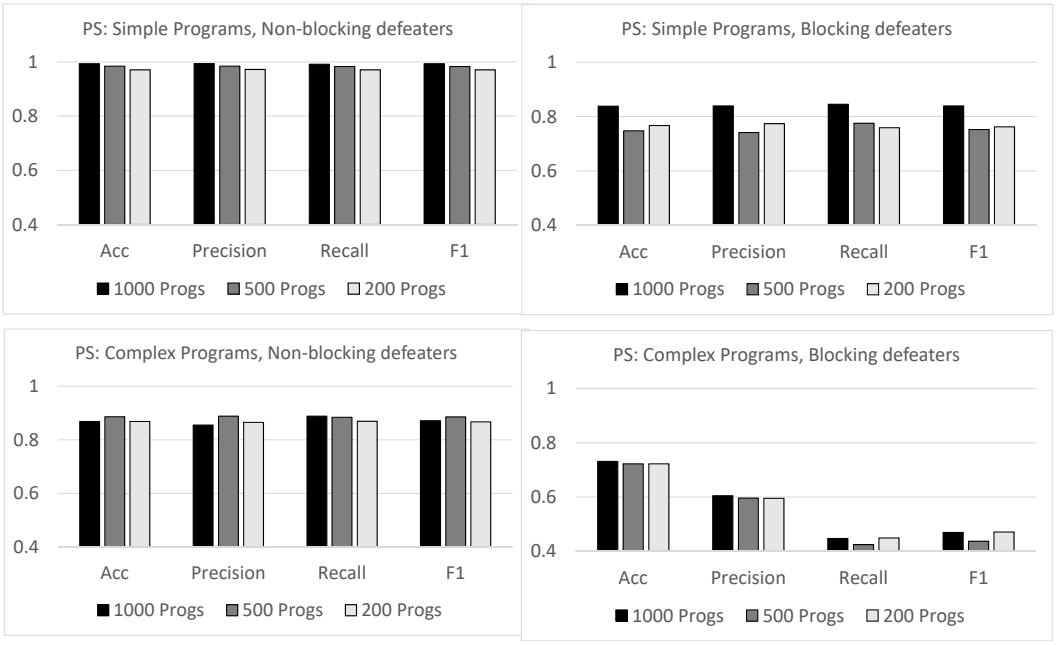

Figure 6: Average performance (Accuracy, Precision, Recall, F1) for the Presumption-enabled Specificity (PS) comparison criterion, varying program complexity (simple, complex), presence of blocking defeaters (not present, present), and number of programs per dataset.

Specificity or Generalized Specificity, and the number of programs from which arguments were extracted. Figure 5 shows an overview of the different datasets considered.

Experimental Setup. For each dataset, a 3-fold cross-validation was done by training three networks with three different random partitions of train and test, respecting a proportion of 80% and 20% respectively.

Results. Figures 6 and 7 show the average Accuracy, Precision, Recall, and F1 scores of the three models along with the standard deviation of the F1 scores for both criteria.

Our results show that a simple neural network architecture with $3,000$ parameters can obtain close to perfect performance in learning two different comparison criteria for arguments of DeLP programs of varying complexity with only proper defeaters by analyzing examples of argument comparisons. If programs have also blocking defeaters, performance is still very good: models achieve high accuracy, precision, recall, and F1 score for datasets generated from simple programs and good accuracy score also for complex programs, though precision, recall, and F1 scores drop, indicating a bias towards the majority class, the blocking defeaters.

Larger datasets created with more programs perform slightly better. Although the improvement is in general very small, it was consistent in all our experiments. Complex programs have worse performance than simple programs only in the presence of blocking defeaters and/or presumptions. Finally, the large difference in performance between blocking and non-blocking datasets could be explained because of the different types of classification done while training with them, using binary vs. multi-class classification.

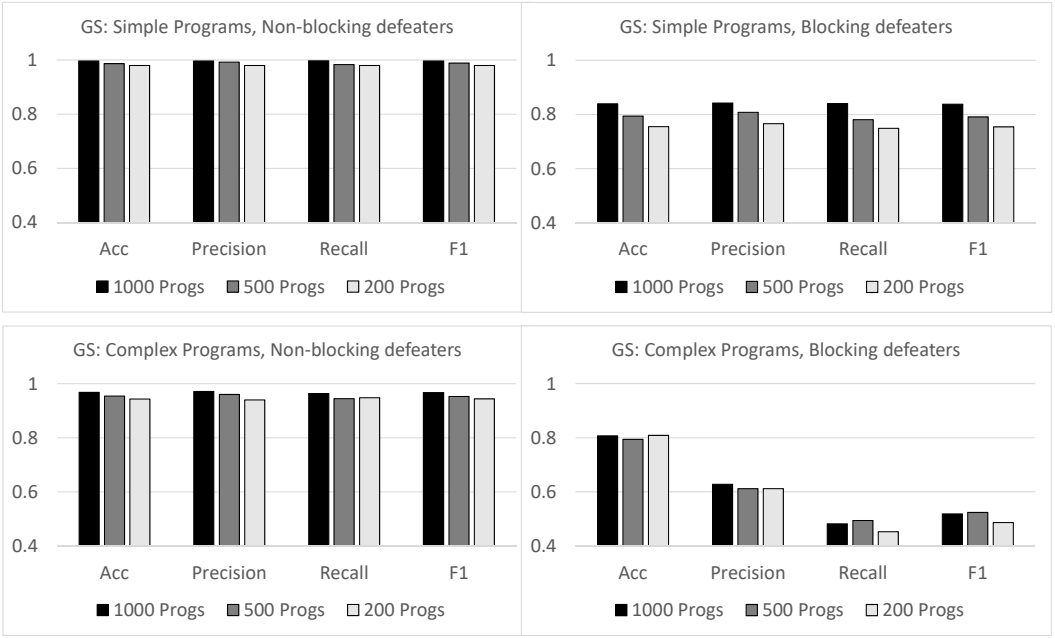

Figure 7: Average performance (Accuracy, Precision, Recall, F1) for the Generalized Specificity (GS) comparison criterion, varying program complexity (simple, complex), presence of blocking defeaters (not present, present), and number of programs per dataset.

5. Related Work

In general terms, this work falls within the area known as *Neuro Symbolic Artificial Intelligence* (NeSy AI) [15, 16], a rich field that encompasses the combination of deep neural networks with symbolic logic for reasoning and learning. Currently, NeSy AI frameworks are capable of carrying out a variety of tasks such as incorporating prior knowledge into deep learning architectures, guiding the learning process with logical constraints, providing symbolic explainability, and using gradient-based approaches to learn logical statements.

More specifically, this paper focuses on *Argumentation and Machine Learning*, an area that aims to integrate the advantages of both fields to address their limitations (for a survey, see [1]). These two fields can greatly benefit from each other: argumentation can improve machine learning in several ways, *e.g.*, by providing it with explainability [17, 18], and machine learning can aid argumentation by building arguments from real data, as in argument mining [19, 20]. The integration of argumentation and machine learning has been proven to be fruitful in different domains, like social network analysis [21], healthcare [22, 17], law [23, 22], and security [24, 25].

Finally, the following line of work is closely related to our own, though it is essentially different in spirit. In [26] an algorithm for translating an agent's knowledge base, expressed as an ODeLP rule base, into a Perceptron-based neural network is introduced. In [27, 28, 29] the authors propose combining a set of criteria specified as a DeLP program with a Fuzzy ART neural network model for solving ambiguities in clustering problems, while [30] also proposes to combine a counter-propagation neural network with a DeLP program for filtering HTML

documents.

6. Conclusions and Future Work

DeLP and neural networks are powerful knowledge representation and problem-solving tools whose combination can be very fruitful. In this preliminary work, we have investigated the feasibility of training a neural network to learn a comparison criterion, obtaining promising results not only for proper defeaters but also when including blocking ones. Particularly, supporting our hypotheses we can say that:

1. Experimental results provide support for Hypothesis 2, since evidence shows that performance of the neural network is affected when the complexity of the program or the comparison criterion increases.
2. With respect of the number of examples being used, we can't confirm that more examples lead to better performance since we obtained similar results on the experiments with only proper defeaters, and the experiments with blocking defeaters showed a decrease in performance with larger datasets.

As future work, we are interested in studying how other, possibly arbitrary and domain-dependent comparison criteria can be learned from data. Towards this end, we are exploring how we can address the general lack of real-world datasets by deriving rules from text in natural language, for which we can take advantage of recent developments in Large Language Models (LLMs). Previous successes in applying DeLP to real-world programs like cyber attribution [24] support working in this direction, where instead of relying on knowledge engineering efforts we might develop general templates that can be instantiated en masse and then, after automatically deriving sets of arguments, these can be labeled based on available ground truth. It will be interesting to analyze whether the classical comparison criteria used in this work come up in such scenarios, or perhaps close variants or entirely new ones arise.

We are also interested in studying possible interactions between the dialectical analysis of arguments that DeLP carries out and neural computation architectures. Finally, it would be interesting to explore combining our models with different attention mechanisms.

ACKNOWLEDGMENTS

This work was supported by funds provided by Universidad Nacional del Sur (UNS) under grants PGI 24/N046 and PGI 24/ZN057), Agencia Nacional de Promoción Científica y Tecnológica under grants PICT-2018-0475 (PRH-2014-0007) and PICT-2020-SERIEA-01481, and CONICET under grants PIP 11220200101408CO and PIP 11220170100871CO. The authors also acknowledge support by the Spanish project PID2022-139835NB-C21 funded by MCIN/AEI/10.13039/501100011033, MCIN/AEI/ 10.13039/501100011033 and by "European Union NextGenerationEU/PRTR".

The authors would like to thank Mario A. Leiva for providing the code for the DeLP program generator that he developed [13], and for his help during its deployment in this project.

References

[1] O. Cocarascu, F. Toni, Argumentation for machine learning: A survey., in: Proc. COMMA, 2016, pp. 219–230.

[2] T. M. Mitchell, T. M. Mitchell, Machine learning, volume 1, McGraw-hill New York, 1997.

[3] K. Gurney, An introduction to neural networks, CRC press, 2018.

[4] J. L. Pollock, Defeasible reasoning, Cognitive science 11 (1987) 481–518.

[5] S. Badreddine, A. d. Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artificial Intelligence 303 (2022) 103649.

[6] R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, et al., Logical neural networks, arXiv preprint arXiv:2006.13155 (2020).

[7] A. J. García, G. R. Simari, Defeasible logic programming: An argumentative approach, Theory and practice of logic programming 4 (2004) 95–138.

[8] M. V. Martinez, A. J. García, G. R. Simari, On the use of presumptions in structured defeasible reasoning, in: Proc. COMMA, IOS Press, 2012, pp. 185–196.

[9] G. R. Simari, R. P. Loui, A mathematical treatment of defeasible reasoning and its implementation, Artificial intelligence 53 (1992) 125–157.

[10] A. J. García, G. R. Simari, Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers, Argument & Computation 5 (2014) 63–88.

[11] F. Stolzenburg, A. J. García, C. I. Chesñevar, G. R. Simari, Computing Generalized Specificity, J. Appl. Non Class. Logics 13 (2003) 87–113.

[12] J. C. Teze, S. Gottifredi, A. J. García, G. R. Simari, An approach to generalizing the handling of preferences in argumentation-based decision-making systems, Knowl. Based Syst. 189 (2020).

[13] M. A. Leiva, Practical Tools for Structured Probabilistic Argumentation with Applications to Cybersecurity (in Spanish), Ph.D. thesis, Departament of Computer Science and Engineering, Universidad Nacional del Sur (UNS), 2022. URL: https://repositoriodigital.uns.edu.ar/handle/123456789/6314.

[14] M. A. Leiva, G. I. Simari, S. Gottifredi, A. J. Garcia, G. R. Simari, DAQAP: defeasible argumentation query answering platform, in: International Conference on Flexible Query Answering Systems, Springer, 2019, pp. 126–138.

[15] P. Shakarian, C. Baral, G. I. Simari, B. Xi, L. Pokala, Neuro Symbolic Reasoning and Learning, Springer, 2023.

[16] P. Hitzler, M. K. Sarker, Neuro-symbolic artificial intelligence: The state of the art (2022).

[17] N. Prentzas, A. Nicolaides, E. Kyriacou, A. Kakas, C. Pattichis, Integrating machine learning with symbolic reasoning to build an explainable AI model for stroke prediction, in: Proc. BIBE, IEEE, 2019, pp. 817–821.

[18] A. Vassiliades, N. Bassiliades, T. Patkos, Argumentation and explainable artificial intelligence: a survey, The Knowledge Engineering Review 36 (2021) e5.

[19] M. Lippi, P. Torroni, Argumentation mining: State of the art and emerging trends, ACM Transactions on Internet Technology (TOIT) 16 (2016) 1–25.

[20] J. Lawrence, C. Reed, Argument mining: A survey, Computational Linguistics 45 (2020) 765–818.

[21] K. Grosse, M. P. Gonzalez, C. I. Chesnevar, A. G. Maguitman, Integrating argumentation and sentiment analysis for mining opinions from Twitter, AI Comm. 28 (2015) 387–401.
[22] M. Možina, J. Žabkar, I. Bratko, Argument based machine learning, Artificial Intelligence 171 (2007) 922–937.
[23] M. Možina, J. Žabkar, T. Bench-Capon, I. Bratko, Argument based machine learning applied to law, Artificial Intelligence and Law 13 (2005) 53–73.
[24] E. Nunes, P. Shakarian, G. I. Simari, A. Ruef, Argumentation models for cyber attribution, in: R. Kumar, J. Caverlee, H. Tong (Eds.), Proc. ASONAM 2016, IEEE Computer Society, 2016, pp. 837–844.
[25] M. Bishop, C. Gates, K. Levitt, Augmenting machine learning with argumentation, in: Proceedings of the New Security Paradigms Workshop, 2018, pp. 1–11.
[26] S. A. Gómez, Modelling derivation in Defeasible Logic Programming with perceptron-based neural networks, in: Proc. CACIC, 2004.
[27] S. A. Gómez, C. I. Chesñevar, Combining argumentation and clustering techniques in pattern classification problems, in: Proc. CACIC, 2003.
[28] S. A. Gómez, C. I. Chesñevar, A hybrid approach to pattern classification using neural networks and defeasible argumentation., in: FLAIRS conference, 2004, pp. 393–398.
[29] S. A. Gómez, C. I. Chesñevar, Integrating defeasible argumentation with fuzzy art neural networks for pattern classification, Journal of Computer Science & Technology, 2004, vol. 4, núm. 1, p. 45-51 (2004).
[30] S. A. Gómez, C. I. Chesñevar, Combining counterpropagation neural networks and defeasible logic programming for text classification, in: Proc. WICC, 2004.