

A view to a KILL: Knowledge Injection via Lambda Layer

Matteo Magnini*, Giovanni Ciatto and Andrea Omicini

Dipartimento di Informatica – Scienza e Ingegneria (DISI), ALMA MATER STUDIORUM—Università di Bologna

Abstract

We propose KILL (Knowledge Injection via Lambda Layer) as a novel method for the injection of symbolic knowledge into neural networks (NN) allowing data scientists to *control* what the network should (not) learn. Unlike other similar approaches, our method does *not* (i) require ground input formulæ, (ii) impose any constraint on the NN undergoing injection, (iii) affect the loss function of the NN. Instead, it acts directly at the backpropagation level, by increasing penalty whenever the NN output is violating the injected knowledge. An experiment is reported to demonstrate the potential (and limits) of our approach.

Keywords

symbolic knowledge injection, AI, ML, neural networks

1. Introduction

A major issue in supervised machine learning (ML) is the usage of *opaque* predictors – such as neural networks (NN) – as black boxes [1]. It is not trivial to understand what NN actually learn from training data, and how they generalise from that data to the whole domain. Currently, state of the art solutions address this issue by supporting their inspection via a plethora of different mechanisms [2].

In this work we tackle the problem from another prospective. We discuss how prior *symbolic* knowledge can be *injected* into NN, during learning, in order to endow them with the designer’s common sense. Injected knowledge may thus harness the NN and make it learn what desired, while preventing it from violating the designer’s constraints—expressed in a symbolic way. Along this line, we propose KILL (Knowledge Injection via Lambda Layer) as a method for the injection of logic formulæ in Datalog form [3] into neural networks of any shape. Unlike other related works, our method (i) does *not* require the input formulæ to be *ground*, (ii) it does not impose any constraint on the NN undergoing injection, (iii) and it does not require the loss function of the network to be affected. Conversely, our method acts directly at the backpropagation level, by increasing the penalty to be backpropagated whenever the NN output

WOA 2022: Workshop “From Objects to Agents”, September 1–2, 2022, Genova, Italy


*Corresponding author.

✉ matteo.magnini@unibo.it (M. Magnini); giovanni.ciatto@unibo.it (G. Ciatto); andrea.omicini@unibo.it (A. Omicini)

🌐 <http://matteomagnini.apice.unibo.it> (M. Magnini); <http://giovanniciatto.apice.unibo.it> (G. Ciatto); <http://andreaomicini.apice.unibo.it> (A. Omicini)

🆔 0000-0001-9990-420X (M. Magnini); 0000-0002-1841-8996 (G. Ciatto); 0000-0002-6655-3869 (A. Omicini)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

is violating the knowledge to be injected. In other words, KILL performs knowledge injection by *constraining* networks' training to adhere to the symbolic knowledge.

To validate our method, we report a simple experiment where the designer's common sense – conveniently represented as logic formulæ – is injected into a NN classifier, to improve its accuracy. Notably, our experiment shows how KILL can be exploited to improve classification performance in inconvenient scenarios where training data is relatively small, and classes are unbalanced and overlapping. Indeed, thanks to symbolic knowledge injection, these inconveniences can be effectively tackled without re-engineering the dataset. The experiment also reveals a lack of robustness w.r.t. extremely rare classes. Notably, this limitation led us to elaborate an interesting discussion on the limits of symbolic knowledge injection attained via constraining.

Accordingly, the paper is organised as follows. Section 2 briefly summarises the background on symbolic knowledge injection, eliciting a number of related works. Then, Section 3 formally describes KILL, as well as its rationale and internal operation. Section 4 then reports our experiments and their design, while results are discussed in Section 5. Finally, Section 6 concludes the paper providing some insights about how the current limitations of KILL could be overcome.

2. Background & related works

In this paper, *symbolic knowledge injection* (SKI) is the task of letting a sub-symbolic predictor exploit formal, symbolic information to improve its predictive performance (e.g. accuracy, F1-measure, learning time, etc.) over data or to use the predictor as a logic engine. Unlike numeric data upon which predictors are commonly trained, symbolic data is generally more compact and expressive, as intensional representations of complex concepts may be concisely written. In particular, symbolic information may encode bold rules that must be satisfied by the concepts the predictor is willing to learn. Hence, provided that some SKI procedure is available, data scientists may craft ad-hoc collections of symbolic expressions aimed at aiding the training of a particular predictor, for a specific learning task. In other words, injection enables provisioning prior knowledge – namely, the designer's common sense – to ML predictors under training.

When it comes to neural networks, approaches for SKI are manifold, and the literature on this topic is vast [4, 5, 6]. Broadly speaking, there exist at least two major sorts of approaches – not mutually exclusive – supporting the injection of symbolic knowledge into neural networks. Approaches of the first sort perform injection during the network's training, using the symbolic knowledge as either a constraint or a guide for the optimisation process [7, 8, 9, 10, 11, 12]. Conversely, approaches of the second sort perform injection by altering the network's architecture to make it mimic the symbolic knowledge [13, 14, 7, 15, 16, 17, 18, 11, 19]. In the remainder of this paper, we focus on approaches of the former sort, as our proposed method falls in this category.

According to the state of the art, the latter strategy commonly works by converting the symbolic knowledge into an additive regularisation term to be added to the loss function used for training. In particular, when the predictor is a NN, knowledge injection is performed during the back propagation step. When the loss function is evaluated, if the network violates the integrated symbolic constraints, then the actual loss value will be greater than the unconstrained case. In

this way, data scientists can lead the networks' learning algorithm to minimise constraints violation, while minimising its error w.r.t. the data as well. Concerning the symbolic knowledge, virtually all techniques we are aware of require information to be represented via (some subset of) first order logic (FOL) formulæ—e.g. propositional logic (quite limited) [14, 7, 15, 18, 9, 10, 12] or full FOL [13, 16, 17]. Actual methods may then vary, depending on (i) which particular sub-set of FOL they rely upon, (ii) how are logic formulæ interpreted as constraints, and (iii) whether formulæ require to be *grounded* or not, before SKI can occur. To the best of our knowledge, virtually all methods proposed so far work by converting formulæ in fuzzy logic functions, and they often require be grounded at some point in the process.

With respect to other state-of-the-art contributions, our proposal differs in several ways. In particular, we accept logic formulæ in Datalog form as input – meaning that we support a restricted subset of FOL –, and we do not require those formulæ to be ground—neither are formulæ grounded anywhere in the process.

3. Knowledge Injection via Lambda Layer

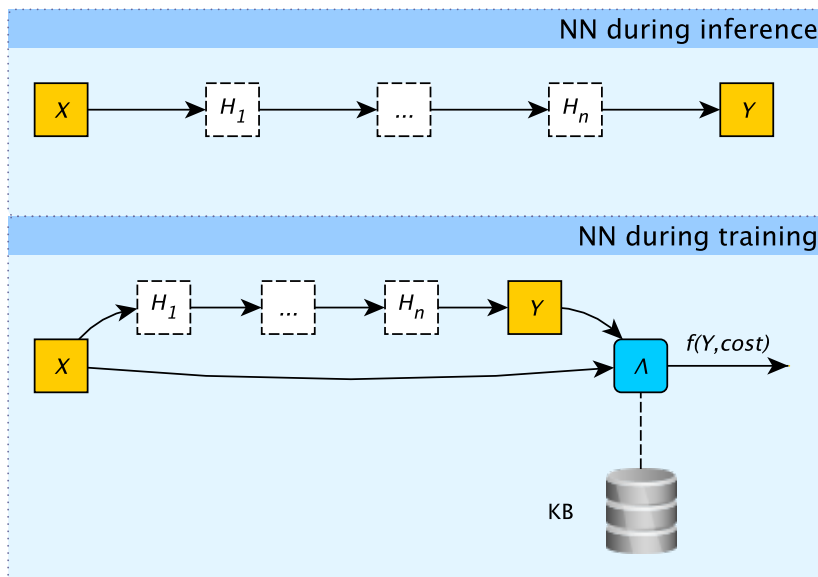
We propose KILL – short for Knowledge Injection via Lambda Layer – as an approach to SKI where the training process is constrained in such a way that the network is penalised whenever its predictions are in conflict with the symbolic knowledge to be injected. In doing so, our approach does not impose any constraint on the architecture (e.g. number of layers, number of neurons, types of activation functions, etc.) or the initialisation status (e.g. random weights or partially trained) of the network subject to injection. It does require, however, (i) the network to have an input and an output layer, and (ii) to be trained via gradient descent. Furthermore, it also requires (iii) the symbolic knowledge to be expressed via one or more Datalog formulæ, and (iv) to encode logic statements about the network's input or output features.

3.1. Λ -layers for SKI

KILL performs injection *during* training. It works by appending one further layer – the Λ -layer henceforth – at the output end of the neural network, and by training the overall network as usual, via gradient descent or others similar strategies. The Λ -layer is in charge of introducing an error (w.r.t. the actual prediction provided by the network's output layer) whenever the prediction violates the symbolic knowledge. The error is expected to affect the gradient descent – or whatever optimisation function – in such a way that violating the symbolic knowledge is discouraged. In other words the NN inductively learns the penalties applied to wrong predictions over the examples and conversely it is much more inclined to avoid such wrong predictions. To serve its purpose, the Λ -layer requires an ad-hoc activation function altering the outcome of the network's original output layer. It also needs the logic formulæ to be numerically interpreted – i.e., converted into functions of real numbers –, to draw actual error values. Once the NN training is over, the injection phase is considered over as well, hence the Λ -layer can be removed and the remaining network can be used as usual. Hence, no architectural property of the original network is hindered by the addition of the Λ -layer.

Differences notwithstanding, injecting by regularising the loss function and injecting using the Λ -layer serve the same purpose. However, in some cases it may be easier to manipulate

Figure 1: General architecture of a NN with n hidden layers, X and Y are respectively the input and output layers. Before training, the KILL algorithm is applied to the predictor so that a new model constrained by the knowledge base (KB) is created. The Λ -layer is kept during the training and then removed for the inference.



the network via the Λ -layer rather than the loss function or in other scenarios the designer may choose to preserve the layer after the training possibly with a different cost function than the one reported in Equation (1). A more detailed discussion about the advantages of Λ -layers is reported in 5.

In the remainder of this section we delve into the details of KILL. First, we discuss how the Λ -layer affects the network architecture, and how it works in detail. Then, we discuss how logic formulæ can be numerically interpreted as errors.

3.2. Additional Λ -layer

We consider the case of a symbolic knowledge base, denoted by \mathcal{K} , to be injected into a feed-forward NN of arbitrary depth, denoted by \mathcal{N} . We denote as X (resp., Y) the input (resp., output) layer of \mathcal{N} . Without loss of generality, we consider the case where the shape of Y is $n \times 1$, but a similar discussion would hold for layers of any shape (e.g. bi-, tri-, or multi-dimensional). We draw no assumptions on the activation function of Y , nor on the amount, topology, or nature of the hidden layers connecting X and Y , nor on the shape of X . Hence, we denote by $\mathbf{y} = [y_1, \dots, y_n]$ the output of Y —i.e., the prediction of the network for some input \mathbf{x} . We also assume \mathcal{K} to be composed by as many rules as the neurons in Y , thus we write $\mathcal{K} = \{\phi_1, \dots, \phi_n\}$, where ϕ_i is a symbolic formula describing/constraining the relation among \mathbf{x} and y_i .

To perform injection, we alter the structure of \mathcal{N} by adding one additional layer – namely, the Λ -layer – as depicted in Figure 1, and we then train it as usual, via gradient descent. The

Λ -layer is densely connected with both X and Y , and its activation function aims at introducing a penalty on y_i every time the formula ϕ_i is violated by some input-output pair (\mathbf{x}, \mathbf{y}) . In particular, we denote as λ the output of the Λ -layer, which is defined as follows:

$$\lambda = \mathbf{y} \times (\mathbf{1} + C(\mathbf{x}, \mathbf{y})) \quad (1)$$

where $C(\mathbf{x}, \mathbf{y})$ is a positive penalty vector representing the cost of modifying the actual output of the network \mathbf{y} .

In turn, the cost vector is defined as follows:

$$C(\mathbf{x}, \mathbf{y}) = [c_1(\mathbf{x}, y_1), \dots, c_i(\mathbf{x}, y_i), \dots, c_n(\mathbf{x}, y_n)] \quad (2)$$

where $c_i : X \times Y \rightarrow [0, 1]$ is a function interpreting ϕ_i as a cost in the $[0, 1]$ range, for possible actual value of \mathbf{x} and y_i .

An in-depth discussion about how logic formulæ can be interpreted as *continuos* (a.k.a. fuzzy) penalties is provided in Section 3.4. For now, it is enough to understand that a penalty is added on the output of the i^{th} neuron of Y whenever the corresponding prediction violates the symbolic knowledge in \mathcal{K} . Such penalty is closer to y_i when the formula ϕ_i is violated the most, while it is closer to 0 either when the formula is violated the less, or there is no formula for that neuron.

The rationale behind the Λ -layer, and the penalty it introduces, is that of altering the output of the network in such a way that its error is very high when it violates the knowledge base to be injected. In this way, the network error is the result of a function of two different components: the actual prediction error and the penalty. The overall error is thus minimised during back propagation, as well as both its components.

3.3. Input knowledge

KILL supports the injection of knowledge bases composed by one or more logic formulæ in “stratified Datalog with negation” form—that is, a variant of Datalog [20] with no recursion (neither direct nor indirect), yet supporting negated atoms.

We choose Datalog because of its expressiveness (strictly higher than propositional logic) and its acceptable limitations. The lack of recursion, in particular, prevents issues when it comes to convert formulæ into neural structures (which are DAG).

More precisely, Datalog is a restricted subset of FOL, representing knowledge via function-free Horn clauses [3]. Horn clauses, in turn, are formulæ of the form $\phi \leftarrow \psi_1 \wedge \psi_2 \wedge \dots$ denoting a logic implication (\leftarrow) stating that ϕ (the head of the clause) is implied by the conjunction among a number of atoms ψ_1, ψ_2, \dots (the body of the clause). Since we rely on Datalog *with negation*, we allow atoms in the bodies of clauses to be negated. In case the i^{th} atom in the body of some clause is negated, we write $\neg\psi_i$. There, each atom $\phi, \psi_1, \psi_2, \dots$ may be a predicate of arbitrary arity.

An l -ary predicate p denotes a relation among l entities: $p(t_1, \dots, t_l)$ where each t_i is a term, i.e., either a constant (denoted in monospace) representing a particular entity, or a logic variable (denoted by *Capitalised Italic*) representing some unknown entity or value. Well-known binary predicates – e.g., $>$, $<$, $=$ – are admissible, too, and retain their usual semantics

from arithmetic. For the sake of readability, we may write these predicates in infix form—hence $> (X, 1) \equiv X > 1$.

Consider for instance the case of a rule aimed at defining when a Poker hand can be classified as a pair—the example may be useful in the remainder of this paper. Assuming that a Poker hand consists of 5 cards, each one denoted by a couple of variables R_i, S_i – where R_i (resp. S_i) is the rank (resp. seed) of the i^{th} card in the hand –, hands of type *pair* may be described via a set of clauses such as the following one:

$$\begin{aligned}
pair(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_1 = R_2 \\
pair(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_2 = R_3 \\
&\vdots \\
pair(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_4 = R_5
\end{aligned} \tag{3}$$

To support injection into a particular NN, we further assume the input knowledge base defines one (and only one) outer relation – say *output* or *class* – involving as many variables as the input and output features the NN has been trained upon. That relation must be defined via one clause per output neuron. Yet, each clause may contain other predicates in their bodies, in turn defined by one or more clause. In that case, since we rely on *stratified* Datalog, we require the input knowledge to *not* include any (directly or indirectly) *recursive* clause definition.

For example, for a 3-class classification task, any provided knowledge base should include a clause such as the following one:

$$\begin{aligned}
class(\bar{X}, y_1) &\leftarrow p_1(\bar{X}) \wedge p_2(\bar{X}) \\
p_1(\bar{X}) &\leftarrow \dots \\
p_2(\bar{X}) &\leftarrow \dots \\
class(\bar{X}, y_2) &\leftarrow p'_1(\bar{X}) \wedge p'_2(\bar{X}) \\
p'_1(\bar{X}) &\leftarrow \dots \\
p'_2(\bar{X}) &\leftarrow \dots \\
class(\bar{X}, y_3) &\leftarrow p''_1(\bar{X}) \wedge p''_2(\bar{X}) \\
p''_1(\bar{X}) &\leftarrow \dots \\
p''_2(\bar{X}) &\leftarrow \dots
\end{aligned}$$

where \bar{X} is a tuple having as many variables as the neurons in the output layer, y_i is a constant denoting the i^{th} class, and $p_1, p_2, p'_1, p'_2, p''_1, p''_2$ are ancillary predicates defined via Horn clauses as well.

3.4. Logic formulæ as penalties

Before undergoing injection, each formula corresponding to some output neuron must be converted into a real-valued function aimed at computing the cost of violating that formula. To this end, we rely on a multi-valued interpretation of logic inspired to Łukasiewicz’s logic [21].

Accordingly, we encode each formula via the $\llbracket \cdot \rrbracket$ function, mapping logic formulæ into real-valued functions accepting real vectors of size $m + n$ as input and returning scalars in \mathbb{R} as output. These scalars are then clipped into the $[0, 1]$ range, via the $\eta : \mathbb{R} \rightarrow [0, 1]$, defined as

Table 1

Logic formulæ's encoding into real-valued functions. There, X is a logic variable, while x is the corresponding real-valued variable, whereas is \bar{X} a tuple of logic variables. Similarly, k is a numeric constant, and k is the corresponding real value, whereas k_i is the constant denoting the i^{th} class of a classification problem. Finally, $\text{expr}(\bar{X})$ is an arithmetic expression involving the variables in \bar{X} .

Formula	C. interpretation	Formula	C. interpretation
$\llbracket \neg \phi \rrbracket$	$\eta(1 - \llbracket \phi \rrbracket)$	$\llbracket \phi \leq \psi \rrbracket$	$\eta(\llbracket \phi \rrbracket - \llbracket \psi \rrbracket)$
$\llbracket \phi \wedge \psi \rrbracket$	$\eta(\max(\llbracket \phi \rrbracket, \llbracket \psi \rrbracket))$	$\llbracket \text{class}(\bar{X}, y_i) \leftarrow \psi \rrbracket$	$\llbracket \psi \rrbracket^*$
$\llbracket \phi \vee \psi \rrbracket$	$\eta(\min(\llbracket \phi \rrbracket, \llbracket \psi \rrbracket))$	$\llbracket \text{expr}(\bar{X}) \rrbracket$	$\text{expr}(\llbracket \bar{X} \rrbracket)$
$\llbracket \phi = \psi \rrbracket$	$\eta(\llbracket \phi \rrbracket - \llbracket \psi \rrbracket)$	$\llbracket \text{true} \rrbracket$	0
$\llbracket \phi \neq \psi \rrbracket$	$\llbracket \neg(\phi = \psi) \rrbracket$	$\llbracket \text{false} \rrbracket$	1
$\llbracket \phi > \psi \rrbracket$	$\eta(0.5 - \llbracket \phi \rrbracket + \llbracket \psi \rrbracket)$	$\llbracket X \rrbracket$	x
$\llbracket \phi \geq \psi \rrbracket$	$\eta(\llbracket \psi \rrbracket - \llbracket \phi \rrbracket)$	$\llbracket k \rrbracket$	k
$\llbracket \phi < \psi \rrbracket$	$\eta(0.5 + \llbracket \phi \rrbracket - \llbracket \psi \rrbracket)$	$\llbracket p(\bar{X}) \rrbracket^{**}$	$\llbracket \psi_1 \vee \dots \vee \psi_k \rrbracket$

* encodes the penalty for the i^{th} neuron

** assuming predicate p is defined by k clauses of the form:
 $p(\bar{X}) \leftarrow \psi_1, \dots, p(\bar{X}) \leftarrow \psi_k$

follows:

$$\eta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 < x < 1 \\ 1 & \text{if } x \geq 1 \end{cases} \quad (4)$$

The resulting values are the penalties discussed in Section 3.2. Hence, the penalty associated with the i^{th} neuron violating rule ϕ_i can be written as $c_i(\mathbf{x}, y_i) = \eta(\llbracket \phi_i \rrbracket(\mathbf{x}, y_i))$.

The $\llbracket \cdot \rrbracket$ encoding function is recursively defined in Table 1. Put it simply, when computing the penalty $c_i(\mathbf{x}, y_i)$ of the i^{th} neuron, KILL looks for the only Datalog rule of the form $\text{class}(\bar{X}, y_i) \leftarrow \psi$. It then focuses on the body of this rule – namely, ψ – ignoring its head—since the head simply reports which expected output the rule is focussing upon. If the body ψ contains some predicates p_1, p_2 , defined by one or more clauses in the provided knowledge base, then these predicates are replaced by the disjunction of the bodies of all clauses defining them. This process is repeated until no ancillary predicates remain in ψ , except for binary expressions involving input variables, constants, arithmetic operators, and logical connectives. Finally, operators and connectives are replaced by continuous functions, as indicated by Table 1. The whole process produces a real-valued interpretation of the original formula which shall be used by KILL to compute $c_i(\mathbf{x}, y_i)$.

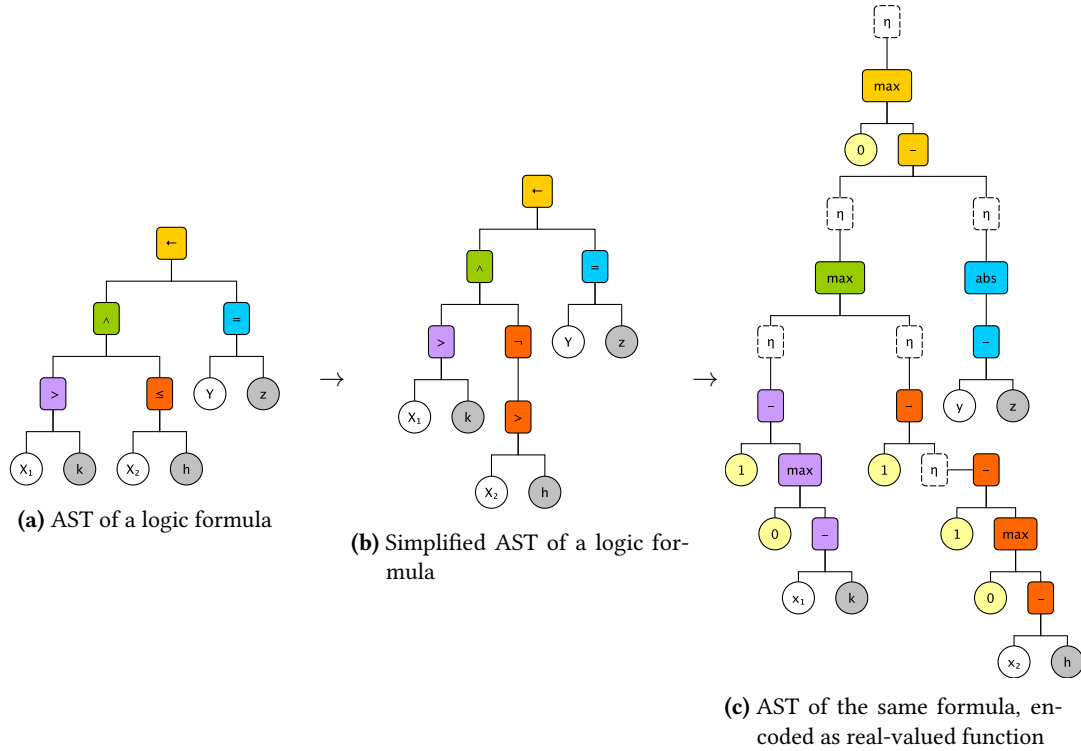
Figure 2 depicts an example of the encoding process where a logic formula is firstly simplified – i.e., converted in a form where it only contains a minimal subset of operators –, and then encoded into an actual real-valued function. The example formula is:

$$\text{class}(X_1, X_2, z) \leftarrow (X_1 \geq k) \wedge (X_2 \geq h)$$

where k, h, z are numeric constants, while X_1 and X_2 are input variables and Y is an output variable. In particular, Figure 2a shows the abstract syntax tree (AST) of this formula, Figure 2b

shows the same AST where the \leq operator is replaced by a *negated* $>$ operator, and Figure 2c shows the AST of the encoded function.

Figure 2: Example of the encoding process of logic formulæ into real-values functions. Only AST are depicted. Box coloured in the same way represent the encoding of a given operator through each encoding step. So, for instance, operator $<$ (red) is firstly converted into a negated \geq , and then in a combination of *max* and subtractions.



4. Experiments

Here we report a number of experiments aimed at assessing KILL for SKI w.r.t. its capability to improve neural networks' predictive performance.¹ A public implementation of the KILL algorithm is available on PSyKI [22].

The design of the experiments is straightforward. We consider a simple learning task – namely, classification – on a finite domain where (i) it is easy to formulate correct constraints in Datalog, and (ii) NN training is difficult because of, e.g., poor separability among classes, as well as unevenly distributed training data. Along this line, we write a set of logic formulæ, one for each class, logically denoting how classification should be performed. We then train a neural network to solve such a classification task, with and without injecting those formulæ. We repeat the experiment by injecting different subsets of formulæ each time. Finally, we assess

¹For the sake of reproducibility, the code of our experiments is available at <https://github.com/MatteoMagnini/kill-experiments-woa-2022>.

Table 2

Poker hand dataset statistics per class.

Class	Train. Instances	Train. Freq. (%)	Test Instances	Test Freq. (%)
nothing	12,493	49.95	501,209	50.12
pair	10,599	42.38	422,498	42.25
two pairs	1,206	4.82	47,622	4.76
three of a kind	513	2.05	21,121	2.11
straight	93	0.37	3,885	0.39
flush	54	0.22	1,996	0.2
full house	36	0.14	1,424	0.14
four of a kind	6	0.024	230	0.023
straight flush	5	0.02	12	0.001
royal flush	5	0.02	3	$1.2 \cdot 10^{-5}$
Total	25,010	100	1,000,000	100

if and under which circumstances SKI succeeds/fails in improving the network’s predictive accuracy.

The rationale behind the experiment design is to assess the effectiveness of SKI in a toy scenario where the correctness of the symbolic knowledge is undoubted, and where an ordinary NN may easily struggle in reaching good predictive performance in a reasonable time. A secondary goal of this design is to identify potential corner cases where SKI falls short.

As we empirically demonstrate in the remainder of this section, KILL is capable of improving the predictive performance of a neural network classifier trained on such dataset, despite being sensitive to severe class unbalancing. A discussion about the possible motivations behind such sensitivity is then provided in Section 5.

4.1. Poker hand data set and logic rules

We rely on the poker hand data set [23], which subtends a multi-classification task on a finite – yet very large – discrete domain, where classes are overlapped and heavily unbalanced, while exact classification rules can be written in logic formulæ. It consists of a tabular dataset, containing 1,025,010 records—each one composed by 11 features. Each record encodes a poker hand of 5 cards. Hence, each records involves 5 couples of features – denoting the cards in the hand –, plus a single categorical feature denoting the class of the hand. Two features are necessary to identify each card: suit and rank. Suit is a categorical feature (heart, spade, diamond and club), while rank is ordinal feature—suitably represented by an integer between 1 and 13 (ace to king). The multi-classification task consists in predicting the poker hand’s class. Each hand may be classified as one of 10 different classes denoting the nature of the hand according to the rules of Poker (e.g. nothing, pair, double pair, flush).

This data set satisfies all the aforementioned requirements: (i) the input space is discrete and finite in size², and the available dataset is just a small sample of it; (ii) classes are extremely unbalanced, as shown in Table 2: a few classes (e.g. nothing and pair) cover nearly half of the

²The size of the input space is the amount 5-permutations of 52 cards, i.e., $\frac{52!}{(52-5)!}$

dataset, while most classes cover less than 1% of the dataset; (iii) there is a hierarchy between classes (e.g. if there are three cards with same rank, then class is three of a kind even if the condition for one pair is satisfied). We use 25,010 records for training and the remaining million for testing, as shown in Table 2.

We define a *class* rule for each class, encoding the preferred way of classifying a Poker hand. For example, let $\{S_1, R_1, \dots, S_5, R_5\}$ be the logic variables representing a Poker hand (S for suit and R for rank), then for class `flush` we define the following rule:

$$\begin{aligned} \text{class}(R_1, S_1, \dots, R_5, S_5, \text{flush}) &\leftarrow \text{flush}(R_1, S_1, \dots, R_5, S_5) \\ \text{flush}(R_1, S_1, \dots, R_5, S_5) &\leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5 \end{aligned} \quad (5)$$

All other rules have the same structure as equation 5: the left-hand side declares the expected class, while the right-hand side describes the necessary conditions for that class—possibly, via some ancillary predicates such as *flush*. Table 3 provides an overview of all the rules we rely upon in our experiments.

4.2. Methodology

It is worth repeating that we choose to use the same data partitioning proposed by the authors of the dataset, meaning that for the training set we rely on 25.010 samples, therefore 1.000.000 for the test set. Such a small training set w.r.t. the test set is quite unusual, since it makes the task more challenging, yet at the same time results are more reliable.

We use the same starting model for all the experiments, consisting of a fully connected NN with 3 layers, where each layer has rectified linear unit (ReLU) as activation function except for the last one that has softmax. We use *categorical cross-entropy* as the loss function for training. After empirical experiments, the best NN has 128 neurons in the first and second layer—10 for the output layer (number of classes). We use batch size equals to 32 and 100 epochs for network’s training. Networks’ performance is evaluated using accuracy, macro-F1 and weighted-F1 score functions. To evaluate network’s performance with knowledge integration we remove the Λ -layer and use the resulting NN as is.

Since we are not relying on any validation set, to avoid overfitting we use 3 stopping criteria during the training of the network: (i) for the 99% of training examples the activation of every output unit is within 0.25 of correct, (ii) at most 100 epochs, (iii) predictor has at least 90% of accuracy on training examples but has not improved its ability to classify training examples for 5 epochs. Similar criteria were used in [14].

Finally, we run 30 experiments for each configuration to have a statistical population for comparisons.

4.3. Results

We define two different configurations for experiments: (i) “classic”, where we use the NN described in Section 4.2, (ii) “knowledge”, where we apply KILL algorithm on the same network architecture. For both configurations we use the same hyper-parameters and run 30 experiments.

Results are reported in Table 4 and in Figure 3.

Table 3

Datalog formulæ describing poker hands. For the sake of readability variables in the head of formulæ and in the arguments of predicates are abbreviated.

Class	Logic Formulation
Pair	$class(R_1, \dots, S_5, \text{pair}) \leftarrow pair(R_1, \dots, S_5)$
	$pair(R_1, \dots, S_5) \leftarrow R_1 = R_2$
	$pair(R_1, \dots, S_5) \leftarrow R_1 = R_3$
	$pair(R_1, \dots, S_5) \leftarrow R_1 = R_4$
	$pair(R_1, \dots, S_5) \leftarrow R_1 = R_5$
	$pair(R_1, \dots, S_5) \leftarrow R_2 = R_3$
	$pair(R_1, \dots, S_5) \leftarrow R_2 = R_4$
	$pair(R_1, \dots, S_5) \leftarrow R_2 = R_5$
	$pair(R_1, \dots, S_5) \leftarrow R_3 = R_4$
	$pair(R_1, \dots, S_5) \leftarrow R_3 = R_5$
Two Pairs	$class(R_1, \dots, S_5, \text{two}) \leftarrow two(R_1, \dots, S_5)$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_4$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_2 = R_4$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_3$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_3 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_3$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_4 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_4$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_4 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_3 = R_5$
	$two(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_3 = R_4$
	$two(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_4 = R_5$
$two(R_1, \dots, S_5) \leftarrow R_2 = R_4 \wedge R_3 = R_5$	
$two(R_1, \dots, S_5) \leftarrow R_2 = R_5 \wedge R_3 = R_4$	
Three of a Kind	$class(R_1, \dots, S_5, \text{three}) \leftarrow three(R_1, \dots, S_5)$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_4$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_5$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_4$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_5$
	$three(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_1 = R_5$
	$three(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_4$
	$three(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_5$
	$three(R_1, \dots, S_5) \leftarrow R_2 = R_4 \wedge R_2 = R_5$
Straight	$class(R_1, \dots, S_5, \text{straight}) \leftarrow royal(R_1, \dots, S_5)$
	$class(R_1, \dots, S_5, \text{straight}) \leftarrow straight(R_1, \dots, S_5)$
	$straight(R_1, \dots, S_5) \leftarrow (R_1 + R_2 + R_3 + R_4 + R_5) = (5 * \min(R_1, \dots, R_5) + 10) \wedge \neg pair(R_1, \dots, S_5)$
Flush	$royal(R_1, \dots, S_5) \leftarrow \min(R_1, \dots, R_5) = 1 \wedge (R_1 + R_2 + R_3 + R_4 + R_5 = 47) \wedge \neg pair(R_1, \dots, S_5)$
	$class(R_1, \dots, S_5, \text{flush}) \leftarrow flush(R_1, \dots, S_5)$
Four of a Kind	$flush(R_1, \dots, S_5) \leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5$
	$class(R_1, \dots, S_5, \text{four}) \leftarrow four(R_1, \dots, S_5)$
	$four(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3 \wedge R_1 = R_4$
	$four(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3 \wedge R_1 = R_5$
	$four(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_4 \wedge R_1 = R_5$
Full House	$four(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_4 \wedge R_1 = R_5$
	$four(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_4 \wedge R_2 = R_5$
Straight Flush	$class(R_1, \dots, S_5, \text{full}) \leftarrow three(\bar{S}_1, \dots, \bar{R}_5) \wedge two(\bar{S}_1, \dots, \bar{R}_5) \wedge \neg four(\bar{S}_1, \dots, \bar{R}_5)$
	$class(R_1, \dots, S_5, \text{straight_flush}) \leftarrow straight(R_1, \dots, S_5) \wedge flush(R_1, \dots, S_5)$
Royal Flush	$class(R_1, \dots, S_5, \text{straight_flush}) \leftarrow royal(R_1, \dots, S_5) \wedge flush(R_1, \dots, S_5)$
Nothing	$class(R_1, \dots, S_5, \text{royal}) \leftarrow royal(R_1, \dots, S_5) \wedge flush(R_1, \dots, S_5)$
	$class(R_1, \dots, S_5, \text{nothing}) \leftarrow \neg pair(R_1, \dots, S_5) \wedge \neg flush(R_1, \dots, S_5) \wedge \neg straight(R_1, \dots, S_5) \wedge \neg royal(R_1, \dots, S_5)$

5. Discussion

In general, experiments show that both predictors are very good in classifying common classes such as “nothing” and “pair”. Also for less frequent classes, “two pairs” and “three of a kind”,

Table 4

Test set accuracy, macro-F1 and weighted-F1 on all classes and single mean class accuracies. All measures represent the mean over the experiment population (30).

Metric	Classic	Knowledge	Metric	Classic	Knowledge
Accuracy	0.962	0.978	Acc. Straight	0.415	0.509
Macro-F1	0.512	0.538	Acc. Flush	0.002	0.002
Weighted-F1	0.96	0.977	Acc. Full	0.628	0.69
Acc. Nothing	0.977	0.989	Acc. Four	0.186	0.19
Acc. Pair	0.968	0.985	Acc. Straight F.	0.003	0
Acc. Two Pairs	0.867	0.914	Acc. Royal F.	0	0
Acc. Three	0.913	0.922			

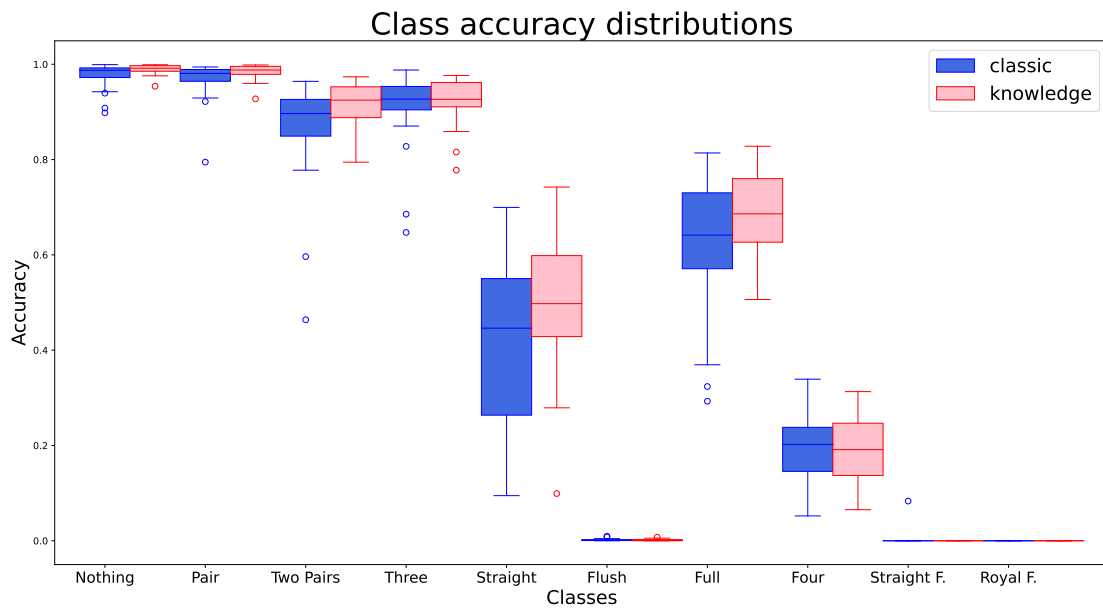


Figure 3: Class accuracy distributions of the two kinds of experiments: blue colour is used for “classic” experiments while red colour is used for the ones with KILL (“knowledge”).

accuracy is still high. The remaining six classes represent the 0.8% of the training set and therefore much more difficult to correctly predict. For “full house” and “straight”, accuracy has middle values, while for the remaining classes accuracy is pretty close to 0. All of this is quite expected due to the strong unbalance in the distribution of classes. The only remarkable fact is that accuracy is extremely low for class “flush” even if less frequent classes such as “full house” and “four of a kind” have higher accuracy. We hypothesise that this happens because the vast majority of data consists in classes which depend only on the values of rank and therefore the network tends to consider it much more. Indeed, only “flush”, “straight flush” and “royal flush” (about the 0.26% of the training set) depend on the values of suit.

Concerning the comparison between the “classic” predictor – with no additional knowledge – and the “knowledge” predictor – obtained by applying KILL algorithm – results show that the

latter has higher performances with statistic significance (Student-T test with p-value < 0.01 when comparing the overall accuracy of the two populations of experiments). In particular, we can observe that the predictor exploiting knowledge during training has on average a higher accuracy for all frequent classes. Also for less common classes such as “straight” and “full house” there is great improvement. However, accuracy is not improved for “flush” and very sporadic classes. We speculate that the reason for the lack of improvement in the prediction of rare classes lies on the kind of injection we are performing. Injecting knowledge by constraining the network during the training is effective as long as classes are represented by a sufficient amount of examples. For instance, if a class is not represented at all, having a logic rule for that specific class is the same as not having it. So, if we have only few units representing a class over a million of examples the effect it is too small to make a difference. We believe that this issue is possibly common upon all SKI algorithms based on constraining, however this should be verified in future works.

To overcome such limit, we may exploit the advantage of having a Λ -layer affecting the constraining instead of operating directly on the loss function of the network. More precisely, we can keep the Λ -layer after the training of the network and just change the function in Equation (1) in such a way that the cost of the violation of the knowledge is used to reduce the network’s error and not to increase it. This should be also explored in future works.

6. Conclusion

In this work we define KILL, a general technique for prior symbolic knowledge injection into deep neural networks. Designers may use Datalog rules to express common sense that are injected through Λ -layer into the network. Rules are encoded into class specific fuzzy logic functions that add a cost to the class prediction value when the rule is violated.

We report a number of experiments where we compare networks without knowledge injection with networks that receives additional information in a multi-classification task. The selected task has some common criticalities of ML classification tasks, in particular data set size limitation, unbalanced classes, class overlapping, and intra-class constraints. Results show that our approach can improve network’s accuracy for classes that are not too sporadic and the overall network’s accuracy. However results reveal a limitation: KILL is quite sensitive w.r.t. situations which have been rarely met during training. We speculate this is general limitation characterising SKI methods acting by constraining the predictor during training. Along this line, further experiments over different methods are required to confirm such general statement.

Accordingly, in our future works we shall consider different Λ -layer functions and test the technique on different domains. To mitigate the sensitivity w.r.t. rare situations, we intend to investigate scenarios where we keep the Λ -layer after the training but instead of increase the network’s error it should reduce the error w.r.t. the prior knowledge (i.e., use a different function in the Λ -layer). In this way we combine techniques based on both constraining during the training and structuring (i.e., altering the predictor’s architecture). This is expected to mitigate the SKI as information about rare situation is encoded into the network structure after the training.

Acknowledgments

This paper was partially supported by the CHIST-ERA IV project “EXPECTATION” – CHIST-ERA-19-XAI-005 –, co-funded by EU and the Italian MUR (Ministry for University and Research).

References

- [1] Z. C. Lipton, The mythos of model interpretability, *Communications of the ACM* 61 (2018) 36–43. doi:10.1145/3233231.
- [2] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, *ACM Computing Surveys* 51 (2019) 93:1–93:42. doi:10.1145/3236009.
- [3] M. Ajtai, Y. Gurevich, Datalog vs first-order logic, *Journal of Computer and System Sciences* 49 (1994) 562–588. doi:10.1016/S0022-0000(05)80071-6.
- [4] T. R. Besold, A. S. d’Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, G. Zaverucha, Neural-symbolic learning and reasoning: A survey and interpretation, in: P. Hitzler, M. K. Sarker (Eds.), *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1–51. doi:10.3233/FAIA210348.
- [5] Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, H. Soh, Embedding symbolic knowledge into deep networks, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019*, pp. 4235–4245. URL: <https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html>.
- [6] R. Calegari, G. Ciatto, A. Omicini, On the integration of symbolic and sub-symbolic techniques for XAI: A survey, *Intelligenza Artificiale* 14 (2020) 7–32. doi:10.3233/IA-190036.
- [7] V. Tresp, J. Hollatz, S. Ahmad, Network structuring and training using rule-based knowledge, in: S. J. Hanson, J. D. Cowan, C. L. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, 1992, pp. 871–878. URL: <http://papers.nips.cc/paper/638-network-structuring-and-training-using-rule-based-knowledge>, NIPS Conference, Denver, Colorado, USA, November 30–December 3, 1992.
- [8] S. Bader, S. Hölldobler, N. C. Marques, Guiding backprop by inserting rules, in: A. S. d. Garcez, P. Hitzler (Eds.), *Proceedings of the 4th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, Patras, Greece, July 21, 2008, volume 366 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2008, pp. 19–22. URL: <http://ceur-ws.org/Vol-366/paper-5.pdf>.
- [9] M. Diligenti, S. Roychowdhury, M. Gori, Integrating prior knowledge into deep learning, in: X. Chen, B. Luo, F. Luo, V. Palade, M. A. Wani (Eds.), *16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, IEEE, 2017, pp. 920–923. doi:10.1109/ICMLA.2017.00-37.
- [10] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. Van den Broeck, A semantic loss function for deep

- learning with symbolic knowledge, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 5498–5507. URL: <http://proceedings.mlr.press/v80/xu18h.html>.
- [11] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data, *Journal of Artificial Intelligence Research* 61 (2018) 1–64. doi:10.1613/jair.5714.
- [12] G. Marra, F. Giannini, M. Diligenti, M. Gori, LYRICS: A general interface layer to integrate logic inference and deep learning, in: U. Brefeld, É. Fromont, A. Hotho, A. J. Knobbe, M. H. Maathuis, C. Robardet (Eds.), *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 283–298. doi:10.1007/978-3-030-46147-8_17.
- [13] D. H. Ballard, Parallel logical inference and energy minimization, in: T. Kehler (Ed.), *Proceedings of the 5th National Conference on Artificial Intelligence*. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science, Morgan Kaufmann, 1986, pp. 203–209. URL: <http://www.aaai.org/Library/AAAI/1986/aaai86-033.php>.
- [14] G. G. Towell, J. W. Shavlik, M. O. Noordewier, Refinement of approximate domain theories by knowledge-based neural networks, in: H. E. Shrobe, T. G. Dietterich, W. R. Swartout (Eds.), *Proceedings of the 8th National Conference on Artificial Intelligence*. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes, AAAI Press / The MIT Press, 1990, pp. 861–866. URL: <http://www.aaai.org/Library/AAAI/1990/aaai90-129.php>.
- [15] A. S. d. Garcez, G. Zaverucha, The connectionist inductive learning and logic programming system, *Applied Intelligence* 11 (1999) 59–77. doi:10.1023/A:1008328630915.
- [16] A. S. d. Garcez, D. M. Gabbay, Fibring neural networks, in: D. L. McGuinness, G. Ferguson (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, July 25-29, 2004, San Jose, California, USA, AAAI Press / The MIT Press, 2004, pp. 342–347. URL: <http://www.aaai.org/Library/AAAI/2004/aaai04-055.php>.
- [17] S. Bader, A. S. d. Garcez, P. Hitzler, Computing first-order logic programs by fibring artificial neural networks, in: I. Russell, Z. Markov (Eds.), *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, Clearwater Beach, Florida, USA, May 15–17, 2005, AAAI Press, 2005, pp. 314–319. URL: <http://www.aaai.org/Library/FLAIRS/2005/flairs05-052.php>.
- [18] M. V. M. França, G. Zaverucha, A. S. d. Garcez, Fast relational learning using bottom clause propositionalization with artificial neural networks, *Machine Learning* 94 (2014) 81–104. doi:10.1007/s10994-013-5392-1.
- [19] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, Neural probabilistic logic programming in deepproblog, *Artificial Intelligence* 298 (2021) 103504. doi:10.1016/j.artint.2021.103504.
- [20] D. Maier, K. T. Tekle, M. Kifer, D. S. Warren, Datalog: concepts, history, and outlook, in: M. Kifer, Y. A. Liu (Eds.), *Declarative Logic Programming: Theory, Systems, and Applications*, ACM / Morgan & Claypool, 2018, pp. 3–100. doi:10.1145/3191315.3191317.
- [21] L. S. Hay, Axiomatization of the infinite-valued predicate calculus, *The Journal of Symbolic Logic* 28 (1963) 77–86. URL: <http://www.jstor.org/stable/2271339>.

- [22] M. Magnini, G. Ciatto, A. Omicini, On the design of PSyKI: a platform for symbolic knowledge injection into sub-symbolic predictors, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), Proceedings of the 4th International Workshop on EXplainable and TRANSPARENT AI and Multi-Agent Systems, volume 13283 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 90–108. doi:10.1007/978-3-031-15565-9_6.
- [23] R. Catral, F. Oppacher, Poker hand data set, UCI machine learning repository, 2007. URL: <https://archive.ics.uci.edu/ml/datasets/Poker+Hand>.