

KINS: Knowledge Injection via Network Structuring

Matteo Magnini^{1,*}, Giovanni Ciatto¹ and Andrea Omicini¹

¹ Dipartimento di Informatica – Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM—Università di Bologna

Abstract

We propose a novel method to inject symbolic knowledge in form of Datalog formulæ into neural networks (NN), called KINS (Knowledge Injection via Network Structuring). The idea behind our method is to extend NN internal structure with ad-hoc layers built out the injected symbolic knowledge. KINS does not constrain NN to any specific architecture, neither requires logic formulæ to be ground. Moreover, it is robust w.r.t. both lack of data and imperfect/incomplete knowledge. Experiments are reported to demonstrate the potential of KINS.

Keywords

neural network, explainable AI, symbolic knowledge injection, KINS, PSyKI

1. Introduction

Supervised machine learning (ML) commonly exploits *opaque* predictors – such as neural networks (NN) – as black boxes [18]. There are several application scenarios where this is becoming troublesome. Indeed, it is non-trivial to forecast what will NN actually learn from data, or whether and how they will grasp general, reusable information for the whole domain. Current state-of-the-art solutions address this issue by supporting a plethora of methods for “opening the black-box” [14]—i.e., inspecting or debugging the inner functioning of NN.

Rather, in this work we tackle the problem of how *injecting* prior *symbolic* knowledge in order to endow them with the designer’s common sense. In this way the issue of opacity is circumvented, as designers may force NN to learn correct-by-design information whenever the situation at hand requires to do so.

Along this line, we propose a novel method for the injection of logic formulæ in Datalog [1] form into NN of arbitrary structure. Our method – called KINS (Knowledge Injection via Network Structuring) – works by extending NN architecture with additional *modules*, i.e., ad-hoc layers reflecting symbolic knowledge. The modules are in charge of numerically computing the truth degree of the logic formulæ to be injected, hence increasing the networks performance in the inference phase. Of course, the network still requires training over data in order to adapt injected knowledge to the particular situation at hand.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy


*Corresponding author.

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

🌐 <https://apice.unibo.it/xwiki/bin/view/MatteoMagnini> (M. Magnini); <https://about.me/gciatto> (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)

Unlike other knowledge injection techniques, KINS (*i*) does not require input formulæ to be *ground*, (*ii*) does not impose any constraint on the NN, and (*iii*) is robust w.r.t. the lack of data exemplifying the injected knowledge. In other words, KINS supports the injection of knowledge describing scenarios where few training data exist. This in turn may let designers suitably handle the case where poor training data covers a given phenomenon the network should be able to deal with—e.g., unbalanced classes in classification tasks.

In order to validate our method, we report an experiment on a well-known benchmark dataset where the designer’s common sense provided by human experts is injected into a NN classifier to improve its performances.

Accordingly, the paper is organised as follows. Section 2 briefly summarises the background on symbolic knowledge injection (SKI). Section 3 formally describes KINS, its rationale and internal operation. Section 4 reports our experiments and their design, whereas results are discussed in Section 5. Finally, Section 6 concludes the paper by providing some insights about how the current limitations of KINS could be overcome.

2. Symbolic Knowledge Injection: Background

We call “symbolic knowledge injection” (SKI) the task of letting a sub-symbolic predictor exploit formal, symbolic information to improve its performances (e.g., accuracy, learning time, need for less training data). Generally speaking, SKI serves the purpose of transferring the designer’s common sense into the predictor, hence overcoming the lack of data, or harnessing predictor towards correct-by-construction directions.

While ML predictors are commonly trained over *numeric* data, formal logic enables representation of knowledge in a compact and expressive way, as intensional representations of complex concepts may be concisely written via logic. Hence, assuming that the input–output relation the ML predictor can learn from data can be expressed in formal logic, and that some SKI procedure is available, human experts may handcraft ad-hoc symbolic knowledge to aid the training of a particular predictor, for a specific learning task. In other words, injection makes it possible to provide ML predictors under training with some prior knowledge.

Many methods for SKI have been proposed into the literature along the years [5, 26, 6]. Most of them target NN for their excellent performances in most ML tasks and domains. Concerning the kind of the provided knowledge, it is virtually always expressed in first order logic (FOL) or subsets of FOL such as Horn’s clauses, Datalog, knowledge graphs and propositional logic. Possible reasons behind these choices are the flexibility of logic in expressing symbolic information, and the malleability of NN—which can be structured in manifold ways to serve disparate purposes.

Broadly speaking, there exist two major sorts of approaches supporting the injection of symbolic knowledge into NN. Methods 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 (i.e., back-propagation). The core idea is to exploit the training step of a NN to increase the error between the prediction value and the expected result when the knowledge is violated. Conversely, approaches of the second sort perform injection by altering the network’s architecture to make it mirror the symbolic knowledge.

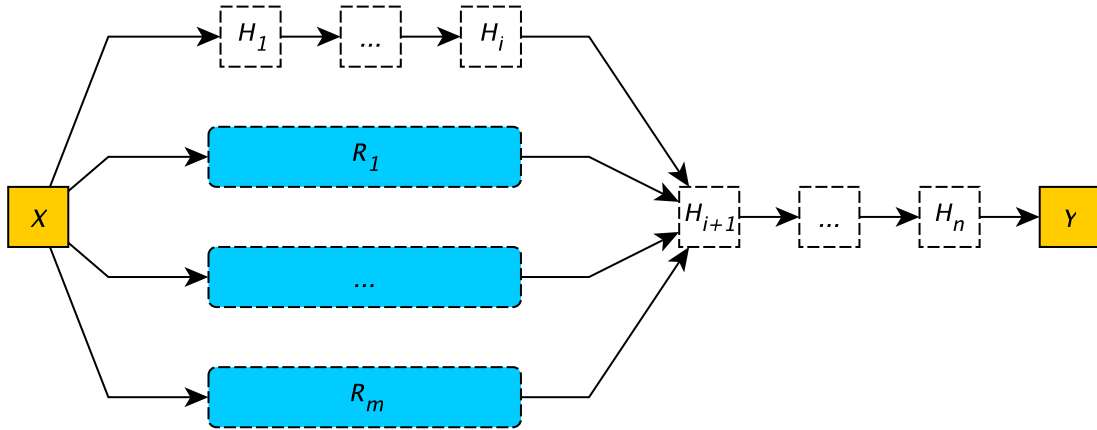


Figure 1: An example of a network’s architecture after the insertion of modules derived from logic formulæ.

One of the first notable works that combine NN and logic rules is KBANN [24]. There, given a set of propositional logic rules, a NN is built by mapping each rule into sub parts of the network. In addition, the loss function of the network is modified with a cost factor that penalises the violation of the prior knowledge—so KBANN exploits both the main injection methods. The algorithm is then validated on classification tasks over biological datasets. In Section 4 our SKI algorithm is compared with KBANN by replicating one of those experiments.

Some other interesting works based on NN structuring are [4, 25, 13, 2, 12, 16, 20], whereas relevant works based on constraining are [3, 7, 9, 10, 27]. In particular, the method in [13] is tested on the same task and with the same methodology as [24]. We obtain similar performance (see Section 5), however they report a greater test accuracy value for KBANN and the other benchmark algorithms w.r.t. [24]: we believe that they consider the best result for each algorithm. More details on SKI algorithms can be found in some recent surveys [5, 26, 6].

3. Injection via Network Structuring

We propose an approach to SKI called *KINS*—short for *Knowledge Injection via Network Structuring*. There, a neural network architecture is extended with additional neural *modules*, structured to reflect and mimic the symbolic knowledge provided by designers. There, a module is a (sub-)network having the same input layer of the original network, yet outputting a value representing the evaluation of a logic formula under a continuous interpretation. The model is aimed at (i) evaluating a specific logic formula against the current input, and (ii) computing the degree of truth of that formula – i.e., a value in range of $[0, 1]$ – to complement the current output. Variables in formulæ are “dynamically grounded” w.r.t. the current input during the feed-forward phase. As a result, non-ground formulæ can be exploited for SKI as such, with no need for any prior groundisation step—which could result unfeasible for non-trivial domains.

It is worth noticing that the provided formulæ are *not* required to cover all possible scenarios. This implies, for instance that rules in classification problems may be provided covering only a

portion of all possible classes.

Figure 1 shows the general architecture of the resulting NN after the injection of m modules (represented as blue rectangles), corresponding to the m rules to be injected. Modules can be arbitrarily complex sub-networks, sharing the same input and their final outputs with the original NN. White boxes represent arbitrary hidden layers H_1, \dots, H_n of the original NN, whereas X is the input layer and Y is the output layer. The injection can be done at any layer H_i and Y . For instance, when dealing with networks that first extract features from the input (such as convolutional NN), then perform classification, one can choose to inject the knowledge in between the two.

Under the hypotheses above, the injection procedure is straightforward. Formulæ are firstly encoded into real-valued functions – hence numerically interpreted –, as described in Section 3.1. Then, a neural module is build to approximate each single real-valued function, following the strategy described in Section 3.2. Finally, that module is added to the original neural network, following the pattern depicted in Figure 1.

Notably, the inner synapses of modules can be either *immutable* – meaning that weights and biases cannot vary during training – or *mutable*—meaning that weights are trainable. Of course, any other synapsis – there including all hidden synapses among layers H_1, \dots, H_i , as well as all the ingoing synapses of layer H_{i+1} and of the following layers – are kept trainable. Thus the NN can exploit both prior knowledge and the information it gathers from data during training. Notice that the synapses connecting each module (and the very last hidden layer) with the output layer are trainable as well. This implies the NN can freely adjust the weights for logic rules during training. The rationale behind this choice is that one cannot assume a logic rule to hold for all the possible patterns in a given domain, yet it may be generally true with a certain degree of confidence. Hence, we let the network learn the relative weight of the injected knowledge w.r.t. the scenario at hand.

In order to operate, KINS does *not* require the loss function to be affected, nor it does 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. So, it can be applied to untrained networks as well as to (partially) trained ones. It does require, however, (i) the network to have an input and an output layer, and (ii) to be trained via gradient descent or similar algorithms. Furthermore, it also requires (iii) symbolic knowledge to be expressed via one or more formulæ in Datalog form, and (iv) logic statements about the network’s input or output features to be encoded.

A public implementation of the algorithm is available as part of the PSyKI framework [19].

3.1. Input Knowledge

KINS supports the injection of knowledge bases composed of one or more logic formulæ in “stratified Datalog with negation” form. Datalog is a restricted subset of first order logic (FOL), representing knowledge via function-free Horn clauses [1]. 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 KINS relies on Datalog *with negation*, atoms in the bodies of clauses are allowed 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 are admissible too, such as $>$, $<$, $=$, etc., which 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 perfect rule (i.e., always true) aimed at defining when a Poker hand can be classified as a pair. 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:

$$\begin{aligned} \text{pair}(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_1 = R_2 \\ \text{pair}(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_2 = R_3 \\ &\vdots \\ \text{pair}(R_1, S_1, \dots, R_5, S_5) &\leftarrow R_4 = R_5 \end{aligned} \tag{1}$$

To support injection into a particular NN, we further assume that input knowledge base defines at least one outer relation – say *output* or *class* – involving as many variables as the input and output features the NN has been trained upon. The relation may be defined via one clause or more, and each clause may possibly leverage on other predicates in their bodies. In turn, each predicate may be defined through 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 instance, for a 3-class classification task, any provided knowledge base should include a clause, as in the following example:

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

where \bar{X} is a tuple having as many variables as the neurons in the output layer, and y_i is a constant denoting the i^{th} class.

3.2. Fuzzy Logic Formulæ as Neural Modules

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 serve this purpose, we rely on a multi-valued interpretation of logic inspired to Łukasiewicz's logic [15] reported in Table 1.

Accordingly, we encode each formula via $\llbracket \cdot \rrbracket$ 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.

Formula	C. interpretation	Formula	C. interpretation
$\llbracket \neg \phi \rrbracket$	$\eta(1 - \llbracket \phi \rrbracket)$	$\llbracket \phi \leq \psi \rrbracket$	$\eta(1 + \llbracket \psi \rrbracket - \llbracket \phi \rrbracket)$
$\llbracket \phi \wedge \psi \rrbracket$	$\eta(\min(\llbracket \phi \rrbracket, \llbracket \psi \rrbracket))$	$\llbracket \text{class}(\bar{X}, y_i) \leftarrow \psi \rrbracket$	$\llbracket \psi \rrbracket^*$
$\llbracket \phi \vee \psi \rrbracket$	$\eta(\max(\llbracket \phi \rrbracket, \llbracket \psi \rrbracket))$	$\llbracket \text{expr}(\bar{X}) \rrbracket$	$\text{expr}(\llbracket \bar{X} \rrbracket)$
$\llbracket \phi = \psi \rrbracket$	$\eta(\llbracket \neg(\phi \neq \psi) \rrbracket)$	$\llbracket \text{true} \rrbracket$	1
$\llbracket \phi \neq \psi \rrbracket$	$\eta(\llbracket \phi \rrbracket - \llbracket \psi \rrbracket)$	$\llbracket \text{false} \rrbracket$	0
$\llbracket \phi > \psi \rrbracket$	$\eta(\max(0, \frac{1}{2} + \llbracket \phi \rrbracket - \llbracket \psi \rrbracket))$	$\llbracket X \rrbracket$	x
$\llbracket \phi \geq \psi \rrbracket$	$\eta(1 + \llbracket \phi \rrbracket - \llbracket \psi \rrbracket)$	$\llbracket k \rrbracket$	k
$\llbracket \phi < \psi \rrbracket$	$\eta(\max(0, \frac{1}{2} + \llbracket \psi \rrbracket - \llbracket \phi \rrbracket))$	$\llbracket p(\bar{X}) \rrbracket^{**}$	$\llbracket \psi_1 \vee \dots \vee \psi_k \rrbracket$

* encodes the value for the i^{th} output

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

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 \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} .

Scalars are then clipped into the $[0, 1]$ range, via function $\eta : \mathbb{R} \rightarrow [0, 1]$ defined as follows:

$$\eta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 < x < 1 \\ 1 & \text{if } x \geq 1 \end{cases} \quad (2)$$

The resulting values are the continuous truth degrees of the formulæ. It is worth noticing that this specific mapping is just one among the many that one may design. Therefore, it could be considered a hyperparameter of the algorithm.

While Table 1 describes the mapping between formulæ and their fuzzy interpretations, we discuss how such an interpretation can be further encoded into neural modules to be added to the NN undergoing injection.

By considering the same domain of Equation (1), we can define a perfect rule for class *flush*, i.e., all cards have the same suit, as follow:

$$\text{class}(\bar{S}, \text{flush}) \leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5 \quad (3)$$

The overall procedure that encodes a logic formula into an ad hoc network is exemplified in Figure 3 – where Equation (3) is converted into a neural module –, and it consists of three phases: (i) the logic formula is parsed and its abstract syntax tree (AST) is constructed, as shown in Figure 3a; (ii) the AST is simplified, merging commutative binary operators, as shown in Figure 3b; and finally (iii) the AST is encoded into a NN, where each operator is converted into a neuron reifying the corresponding operation specified in Table 1, as shown in Figure 3c. In particular, in the last step, operators are converted by recursively applying the encoding rules graphically defined in Figure 2. There, variables S_i are mapped into input neurons, while constants possibly occurring in formulæ are mapped into neurons with constant

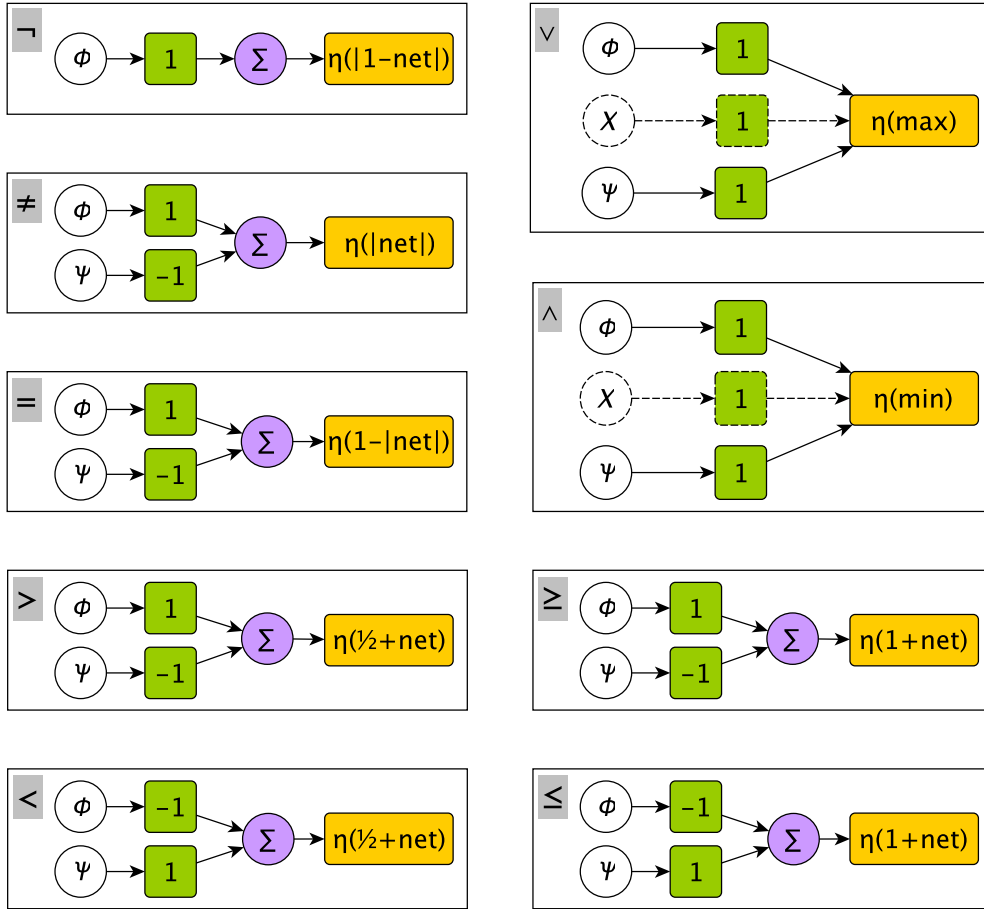


Figure 2: Mapping of formulæ into neurons. White circles are input variables (I), green boxes represent the corresponding weights (W), purple circles are the sum of the weighted inputs ($W \times I$). Yellow rectangles are activation functions, net is the output of $W \times I$, max and min respectively the maximum and minimum of input values, η is the function described in Equation (2).

output. Similarly, algebraic operators such as addition and multiplication are encoded in single neurons that perform the same operation.

4. Experiments

Here we report experiments aimed at assessing KINS for SKI w.r.t. its capability to improve NN's predictive performance. For the sake of reproducibility, the code of our experiments is available at <https://github.com/matteomagnini/kins-experiments-cilc-2022>.

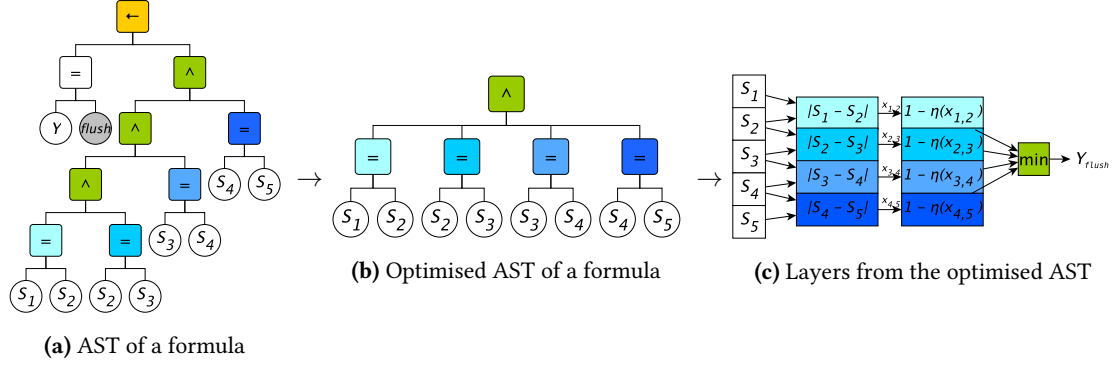


Figure 3: Example of the encoding process of formulæ into module network. Box coloured in the same way represent the encoding of a given operator through each encoding step.

4.1. Primate Splice-Junction Gene Sequences

To validate our method, we test KINS performance on a well-known benchmark: the primate splice-junction gene sequences (PSJGS) dataset [11]. The dataset consists of 3190 records, each of them represents a sequence of 60 DNA nucleotides—namely adenine (a), cytosine (c), guanine (g) and thymine (t). Each sequence starts from position -30 up to 30, zero excluded. One DNA sequence can be classified as an intron–exon (ie) boundary, an exon–intron (ei) boundary, or none (n) of them. Class frequencies are 50% for n, 25% for both ie and ei.

The PSJGS dataset comes with a set of textual logical rules aimed at classifying DNA sequences provided by human experts. In Table 2 we report the same rules converted in Datalog form. Datalog rules are equivalent to the original ones that are expressed in a different custom formalism, but they are machine-interpretable as well.

Within Datalog rules, variables are indexed starting from -30 to 30, zero excluded: $X_{-30}, \dots, X_{-1}, X_{+1}, \dots, X_{+30}$. There, variable $X_{\pm i}$ denotes the value of the nucleotide in position $\pm i$, which is represented via ad-hoc constants (namely, a, c, g, t). For the sake of readability, we write \bar{X} in place of the full sequence of variables X_{-30}, \dots, X_{+30} .

It is worth noticing that the original rules from the PSJGS dataset include different symbols to denote multiple possible nucleotides in a compact way. Table 3 reports the meaning of the additional symbols: rules in Table 2 are reported using them.

When classifying data from the PSJGS dataset according to the rules in Table 2, sequences of type ie are correctly classified 295 times – true positives (TP) –, however the rule is also true for 25 ei records and for 3 n records—false positives (FP). Instead, ei sequences are correctly classified 31 times, and there are no FP. Figure 4 shows the confusion matrix of the rules considering also a fictional rule for class n that corresponds to the logical *and* of both ie and ei rules negated:

$$\text{class}(\bar{X}, n) \leftarrow \neg \text{class}(\bar{X}, ei) \wedge \neg \text{class}(\bar{X}, ie) \quad (4)$$

While this is far from being perfect knowledge describing the entire domain with no or few errors, it is still good enough to positively affect the training of the predictor.

Class	Logic Formulation
EI	$\begin{aligned} \text{class}(\bar{X}, \text{ei}) \leftarrow & X_{-3} = \text{m} \wedge X_{-2} = \text{a} \wedge X_{-1} = \text{g} \wedge X_{+1} = \text{g} \wedge \\ & X_{+2} = \text{t} \wedge X_{+3} = \text{a} = \text{r} \wedge X_{+4} = \text{a} \wedge \\ & X_{+5} = \text{g} \wedge X_{+6} = \text{t} \wedge \neg(\text{ei_stop}(\bar{X})) \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-3} = \text{t} \wedge X_{-2} = \text{a} \wedge X_{-1} = \text{a} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-3} = \text{t} \wedge X_{-2} = \text{a} \wedge X_{-1} = \text{g} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-3} = \text{t} \wedge X_{-2} = \text{g} \wedge X_{-1} = \text{a} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-4} = \text{t} \wedge X_{-3} = \text{a} \wedge X_{-2} = \text{a} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-4} = \text{t} \wedge X_{-3} = \text{a} \wedge X_{-2} = \text{g} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-4} = \text{t} \wedge X_{-3} = \text{g} \wedge X_{-2} = \text{a} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-5} = \text{t} \wedge X_{-4} = \text{a} \wedge X_{-3} = \text{a} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-5} = \text{t} \wedge X_{-4} = \text{a} \wedge X_{-3} = \text{g} \\ \text{ei_stop}(\bar{X}) \leftarrow & X_{-5} = \text{t} \wedge X_{-4} = \text{g} \wedge X_{-3} = \text{a} \end{aligned}$
	$\begin{aligned} \text{class}(\bar{X}, \text{ie}) \leftarrow & \text{pyramidine_rich}(\bar{X}) \wedge \neg(\text{ie_stop}(\bar{X})) \wedge \\ & X_{-3} = \text{y} \wedge X_{-2} = \text{a} \wedge X_{-1} = \text{g} \wedge X_{+1} = \text{g} \\ \text{pyramidine_rich}(\bar{X}) \leftarrow & 6 \leq (X_{-15} = \text{y} + \dots + X_{-6} = \text{y}) \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+2} = \text{t} \wedge X_{+3} = \text{a} \wedge X_{+4} = \text{a} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+2} = \text{t} \wedge X_{+3} = \text{a} \wedge X_{+4} = \text{g} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+2} = \text{t} \wedge X_{+3} = \text{g} \wedge X_{+4} = \text{a} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+3} = \text{t} \wedge X_{+4} = \text{a} \wedge X_{+5} = \text{a} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+3} = \text{t} \wedge X_{+4} = \text{a} \wedge X_{+5} = \text{g} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+3} = \text{t} \wedge X_{+4} = \text{g} \wedge X_{+5} = \text{a} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+4} = \text{t} \wedge X_{+5} = \text{a} \wedge X_{+6} = \text{a} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+4} = \text{t} \wedge X_{+5} = \text{a} \wedge X_{+6} = \text{g} \\ \text{ie_stop}(\bar{X}) \leftarrow & X_{+4} = \text{t} \wedge X_{+5} = \text{g} \wedge X_{+6} = \text{a} \end{aligned}$

Table 2

Datalog formulæ describing DNA classification criteria generated from the original one.

4.2. Methodology

To make our experiments comparable with already existing literature benchmarks, we follow the very same method used by Towell and Shavlik in [24]. We use 10-fold cross validation with a training size of 1000 randomly-chosen records—drawn among the 3190 available ones (i.e., 31.3% of the overall dataset). Then, for each fold, we train one instance of KINS. Finally, test accuracy is computed on the 2190 records excluded from training, by averaging the predictions of the 10 KINS instances. Unlike the original method, we repeat the overall experiment 30 times – instead of just 10 – to improve result significance.

Symbol	Adenine	Cytosine	Guanine	Thymine	Logic form
d	•		•	•	$(X_i = d) \equiv (X_i = a \vee X_i = g \vee X_i = t)$
m	•	•			$(X_i = m) \equiv (X_i = a \vee X_i = c)$
r	•		•		$(X_i = r) \equiv (X_i = a \vee X_i = g)$
s		•	•		$(X_i = s) \equiv (X_i = c \vee X_i = g)$
y		•		•	$(X_i = y) \equiv (X_i = c \vee X_i = t)$

Table 3

Mapping of aggregative symbols and the four nucleotides. Each symbol can be substituted with one base on the right that has a dot.

More precisely, each time we train an instance of KINS we leverage on a NN with 3 fully connected layers: input layer (60 neurons), hidden layer (neurons), and output layer (3 neurons). During training, we exploit dropout [23] for each layer, up to some extent (0.2), to increase robustness of the network w.r.t. overfitting. Layers have rectified linear unit as activation function, except the output one that has Softmax. The optimiser used for training is Adam [17], categorical cross-entropy as loss function. We use the same stopping criteria used in [24], namely: (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.

Rules (Table 2) $ei_stop(\bar{X})$ and $ie_stop(\bar{X})$ are immutable, while $class(\bar{X}, ei)$, $class(\bar{X}, ie)$ and $pyrimidine_rich(\bar{X})$ are mutable. We recall that mutable rules have trainable weights whereas immutable rules have fixed weights—structure is always preserved.

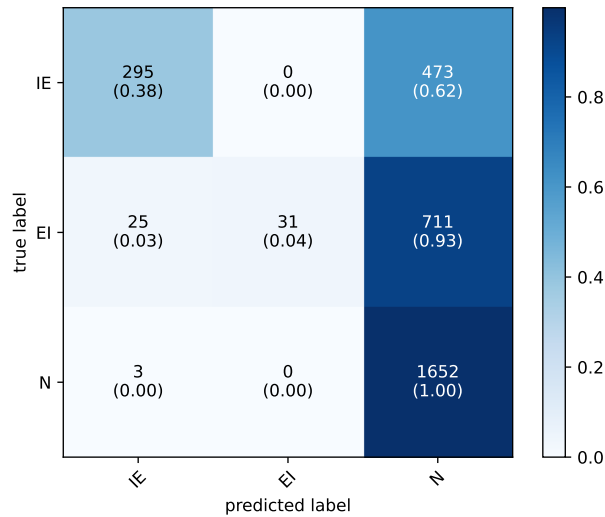


Figure 4: Confusion matrix using only the provided knowledge to classify DNA sequences.

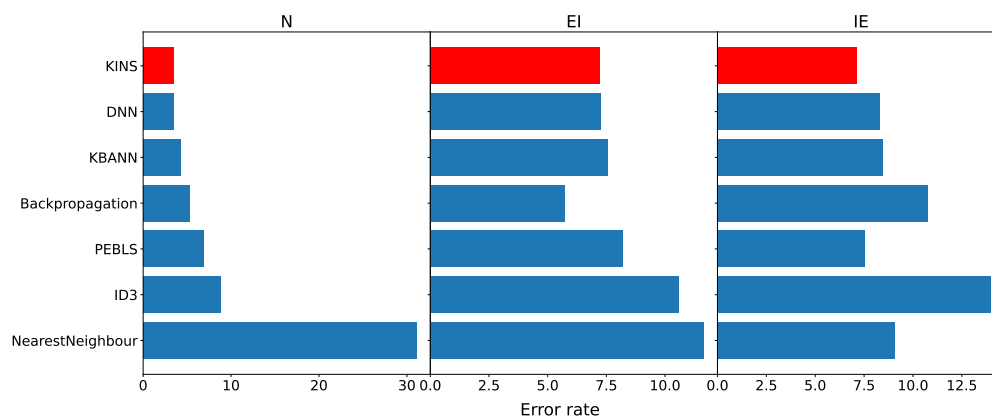


Figure 5: Per-class error rate of different algorithms on the primate slice-junction gene sequences dataset.

5. Discussion

We test KINS by injection the prior knowledge in different layers of the NN previously described. Best results are obtained when the injection is performed at the first hidden layer. Following the aforementioned methodology, we have 10 predictors trained on 900 records each – 1000 unique training records in total out of 3190 – for each experiment. We run 30 experiments using 10-fold cross validation with random weights initialisation.

After the injection of the prior knowledge into the first layer and training, the mean accuracy of the 30 experiment on the test set is 94.73%. Single mean class accuracies are: (ie) 92.79%, (ei) 92.49%, (n) 96.67%.

We execute 30 additional runs using the same base architecture NN *without* the injection of any knowledge obtaining the following results: (mean accuracy) 94.45%, (ie) 91.67%, (ei) 92.73%, (n) 96.54%. After computing Student’s T-test on the two distributions we reject the null hypothesis: predictors generated from KINS have better accuracies with statistic relevance.

The improvement of the accuracy using our injection method is significant even with imperfect knowledge. Figure 5 reports the error rate per single class using different algorithms. KINS is our knowledge injection method, while DNN is the network used in KINS, but without knowledge injection. KBANN is the algorithm proposed in [24]: it performs slightly worse than DNN and KINS. Arguably, the main reason for this difference in performance is that the entire structure of KBANN reflects the provided knowledge, whereas in KINS a portion of the network is free to adapt to the data. This is a strength when the knowledge is close to the real rules for the domain, but clearly a weakness in the opposite scenario. The remaining algorithms are (i) standard back-propagation [22], (ii) PEBLS [8], (iii) ID3 [21], and (iv) nearest neighbours. Generally, they all perform worse than KINS.

6. Conclusion

In this work we define KINS, a general technique for symbolic knowledge injection into deep neural networks. Designer uses rules in Datalog form (stratified with negation) to express common sense, which are injected through additional modules – ad-hoc layers – capable of evaluating the truth degree of the rules themselves. Rules are interpreted as class-specific fuzzy-logic functions that are then used to build the modules to be inserted into the NN.

We report a number of experiments where we compare networks without knowledge injection with networks – architecturally equivalent except for knowledge injection – that receives additional information in a multi-classification task. We also compare our method with different algorithms, in particular KBANN, which is also based on knowledge injection. The selected task has some of the common criticalities of ML classification tasks, in particular data set size limitation and unbalanced classes. Moreover, the provided prior knowledge is far to be perfect. Results show that our approach can improve network’s accuracy with statistical significance.

Investigating the joint use of SKI and symbolic knowledge extraction (SKE) in the same ML workflow is indeed a topic of major interest, which we plan to explore. Introducing multiple cycles of SKI and SKE, possibly using different kind of predictors, could bring several benefits (e.g., final performances of the predictor, more precise knowledge).

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] Ajtai, M., Gurevich, Y.: Datalog vs first-order logic. *Journal of Computer and System Sciences* **49**(3), 562–588 (1994). [https://doi.org/10.1016/S0022-0000\(05\)80071-6](https://doi.org/10.1016/S0022-0000(05)80071-6)
- [2] Bader, S., Garcez, A.S.d., Hitzler, P.: Computing first-order logic programs by fibring artificial neural networks. In: Russell, I., Markov, Z. (eds.) *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, Clearwater Beach, Florida, USA, May 15–17, 2005. pp. 314–319. AAAI Press (2005), <http://www.aaai.org/Library/FLAIRS/2005/flairs05-052.php>
- [3] Bader, S., Hölldobler, S., Marques, N.C.: Guiding backprop by inserting rules. In: Garcez, A.S.d., Hitzler, P. (eds.) *Proceedings of the 4th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, Patras, Greece, July 21, 2008. *CEUR Workshop Proceedings*, vol. 366. CEUR-WS.org (2008), <http://ceur-ws.org/Vol-366/paper-5.pdf>
- [4] Ballard, D.H.: Parallel logical inference and energy minimization. In: Kehler, T. (ed.) *Proceedings of the 5th National Conference on Artificial Intelligence*. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science. pp. 203–209. Morgan Kaufmann (1986), <http://www.aaai.org/Library/AAAI/1986/aaai86-033.php>
- [5] Besold, T.R., d’Avila Garcez, A.S., Bader, S., Bowman, H., Domingos, P.M., Hitzler, P., Kühnberger, K., Lamb, L.C., Lowd, D., Lima, P.M.V., de Penning, L., Pinkas, G., Poon, H.,

- Zaverucha, G.: Neural-symbolic learning and reasoning: A survey and interpretation. CoRR **abs/1711.03902** (2017), <http://arxiv.org/abs/1711.03902>
- [6] Calegari, R., Ciatto, G., Omicini, A.: On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale* **14**(1), 7–32 (2020). <https://doi.org/10.3233/IA-190036>
- [7] Che, Z., Kale, D.C., Li, W., Bahadori, M.T., Liu, Y.: Deep computational phenotyping. In: Cao, L., Zhang, C., Joachims, T., Webb, G.I., Margineantu, D.D., Williams, G. (eds.) *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (KDD)*, Sydney, NSW, Australia, August 10-13, 2015. pp. 507–516. ACM (2015). <https://doi.org/10.1145/2783258.2783365>
- [8] Cost, S., Salzberg, S.: A weighted nearest neighbor algorithm for learning with symbolic features. *Machine learning* **10**(1), 57–78 (1993)
- [9] Demeester, T., Rocktäschel, T., Riedel, S.: Lifted rule injection for relation embeddings. In: Su, J., Carreras, X., Duh, K. (eds.) *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Austin, Texas, USA, November 1-4, 2016. pp. 1389–1399. The Association for Computational Linguistics (2016). <https://doi.org/10.18653/v1/d16-1146>
- [10] Diligenti, M., Roychowdhury, S., Gori, M.: Integrating prior knowledge into deep learning. In: Chen, X., Luo, B., Luo, F., Palade, V., Wani, M.A. (eds.) *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cancun, Mexico, December 18-21, 2017. pp. 920–923. IEEE (2017). <https://doi.org/10.1109/ICMLA.2017.00-37>
- [11] Dua, D., Graff, C.: UCI machine learning repository (2017), <http://archive.ics.uci.edu/ml>
- [12] França, M.V.M., Zaverucha, G., Garcez, A.S.d.: Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning* **94**(1), 81–104 (2014). <https://doi.org/10.1007/s10994-013-5392-1>
- [13] d’Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive learning and logic programming system. *Appl. Intell.* **11**(1), 59–77 (1999). <https://doi.org/10.1023/A:1008328630915>
- [14] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. *ACM Computing Surveys* **51**(5), 93:1–93:42 (2019). <https://doi.org/10.1145/3236009>
- [15] Hay, L.S.: Axiomatization of the infinite-valued predicate calculus. *The Journal of Symbolic Logic* **28**(1), 77–86 (1963), <http://www.jstor.org/stable/2271339>
- [16] Hu, Z., Ma, X., Liu, Z., Hovy, E.H., Xing, E.P.: Harnessing deep neural networks with logic rules. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, Berlin, Germany, August 7-12, 2016. The Association for Computer Linguistics (2016). <https://doi.org/10.18653/v1/p16-1228>
- [17] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), <http://arxiv.org/abs/1412.6980>
- [18] Lipton, Z.C.: The mythos of model interpretability. *Communications of the ACM* **61**(10), 36–43 (2018). <https://doi.org/10.1145/3233231>
- [19] Magnini, M., Ciatto, G., Omicini, A.: On the design of PSyKI: A platform for symbolic knowledge injection into sub-symbolic predictors. In: Calvaresi, D., Najjar, A., Winikoff,

- M., Främling, K. (eds.) Explainable and Transparent AI and Multi-Agent Systems – Fourth International Workshop, EXTRAAMAS 2022, Virtual Event, May 9-10, 2021, Revised Selected Papers. Lecture Notes in Computer Science, Springer (2022)
- [20] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., De Raedt, L.: Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence* **298**, 103504 (2021). <https://doi.org/10.1016/j.artint.2021.103504>
- [21] Quinlan, J.R.: Induction of decision trees. *Machine Learning* **1**(1), 81–106 (1986). <https://doi.org/10.1007/BF00116251>
- [22] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L.L., PDP Research Group (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations. Bradford Books (1985), <https://mitpress.mit.edu/books/parallel-distributed-processing-volume-1>
- [23] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014), <http://dl.acm.org/citation.cfm?id=2670313>
- [24] Towell, G.G., Shavlik, J.W.: Knowledge-based artificial neural networks. *Artif. Intell.* **70**(1-2), 119–165 (1994). [https://doi.org/10.1016/0004-3702\(94\)90105-8](https://doi.org/10.1016/0004-3702(94)90105-8)
- [25] Tresp, V., Hollatz, J., Ahmad, S.: Network structuring and training using rule-based knowledge. In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) *Advances in Neural Information Processing Systems 5*, NIPS 1992, Denver, Colorado, USA, November 30 – December 3, 1992. pp. 871–878. Morgan Kaufmann (1992), <http://papers.nips.cc/paper/638-network-structuring-and-training-using-rule-based-knowledge>
- [26] Xie, Y., Xu, Z., Meel, K.S., Kankanhalli, M.S., Soh, H.: Embedding symbolic knowledge into deep networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E.B., Garnett, R. (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. pp. 4235–4245 (2019), <https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html>
- [27] Xu, J., Zhang, Z., Friedman, T., Liang, Y., Van den Broeck, G.: A semantic loss function for deep learning with symbolic knowledge. In: Dy, J.G., Krause, A. (eds.) *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. *Proceedings of Machine Learning Research*, vol. 80, pp. 5498–5507. PMLR (2018), <http://proceedings.mlr.press/v80/xu18h.html>