Knowledge injection of Datalog rules via Neural Network Structuring with KINS

Knowledge injection of Datalog rules via Neural Network Structuring with KINS*

Matteo Magnini[1 [0000−0001−9990−420X]], Giovanni Ciatto[1 [0000−0002−1841−8996]], and Andrea Omicini[1 [0000−0002−6655−3869]]

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna
{matteo.magnini, giovanni.citatto, andrea.omicini}@unibo.it

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 of 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, involving multiple datasets and predictor types, to demonstrate how KINS can significantly improve the predictive performance of the neural networks it is applied to.

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 [22]. There are several application scenarios where this is becoming troublesome. Indeed, it is non-trivial to forecast what NN would actually learn from data, or whether and how they would 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" [18]—i.e., inspecting or debugging the inner functioning of NN.

In this work we tackle instead the problem of how *injecting* prior *symbolic* knowledge in order to endow them with the designer's common sense. This circumvents the issue of opacity, as designers can force NN to learn correct-by-design information whenever required.

Along this line, we propose a novel method for the injection of logic formulæ in Datalog form [1] into arbitrarily-structured NN. 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

* This paper is an extended version of [23]

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 specific circumstances.

Unlike other knowledge-injection techniques, KINS *(i)* does not require input formulæ to be *ground*, *(ii)* imposes no constraints 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 cover 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 a number of experiments on well-known benchmark datasets—namely, the primate splice-junction gene sequences dataset and the Wisconsin breast cancer dataset [13]. There, the designer's common sense provided by human experts is injected into neural networks classifiers to improve their predictive performances. Student-T tests are reported as well, to demonstrate significance of injection, as perfomed by KINS. Notably, experiments are conducted by leveraging on the PSyKI technology for symbolic knowledge injection (SKI) algorithms [24].

Accordingly, the paper is organised as follows. Section 2 briefly summarises the background on 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 representation of complex concepts can be concisely written in some logic formalism. 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 so as 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 [6, 8, 33]. 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—namely, *constraining* and *structuring*. Methods of the first sort perform injection during the network training, using 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 *structure* to make it mirror the symbolic knowledge.

Some interesting works based on NN structuring are [4, 14–16, 20, 26, 29, 30] There, one can notice a strong trend towards propositional logic (see [14, 16, 20, 26, 29, 30]) instead of other logic formalisms. The main reason for that is probably the very nature of propositional logic and NN. In fact, propositional logic is quite simple, it does not allow any type of recursion mechanism and quantifiers, terms and non-atomic predicates are missing, therefore it is easy to manipulate. Modern NN make use of the back-propagation-based algorithms to be trained. This means that in the computational graph of a NN cycles are not allowed. As a result, the integration of recursive logic is not trivial at all. One notable attempt to address FOL has been made in [15] where the authors define fibred NN—i.e., NN that encapsulates other NN and are simultaneously trained. In [2] authors show how a fibred NN can be used to solve a recursive problem. Whereas relevant works based on constraining are [3, 9, 11, 12, 34]. Usually SKI constraining-based algorithms add a cost factor to the loss function. The cost factor depends on the provided knowledge and in general it is true that the higher the cost the higher the predictor violates the knowledge. More details on SKI algorithms can be found in some recent surveys [6, 8, 33].

Since our intention is to validate the algorithm that we propose in Section 3 with other predictors, we choose to compare it with one of the SKI algorithm from the literature. In the following section we provide a more detailed description on how KBANN [29] works.

2.1 KBANN

KBANN is one of the first notable works combining NN and logic rules. 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 may be modified with a cost factor that penalises the violation of the prior knowledge—so KBANN could exploit both 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.

The main KBANN algorithm builds from scratch a NN starting from a set of propositional clauses: logic operators are mapped into neurons and logic variables are mapped into connections. The author provides a set of instructions that constitutes the rules-to-network algorithm—in particular, a mapping for the *conjunction*, *disjunction*, and *negation* logic operators. Weights, biases, and activation functions of the neurons are initialised during the construction of the NN according to the mapping. A perturbation phase can follow before training to avoid problems related to symmetry. It is worth noting that weights and activation functions of the neurons depend on a hyper-parameter ω.

Additionally, KBANN may include a cost factor in the loss function. This additional factor is computed as the sum of the absolute differences between the starting weights and the current weights of the network. The rationale is that the more the NN diverges from the starting configuration – which is assumed to reflect the provided knowledge – the higher is the cost.

3 Injection via Network Structuring

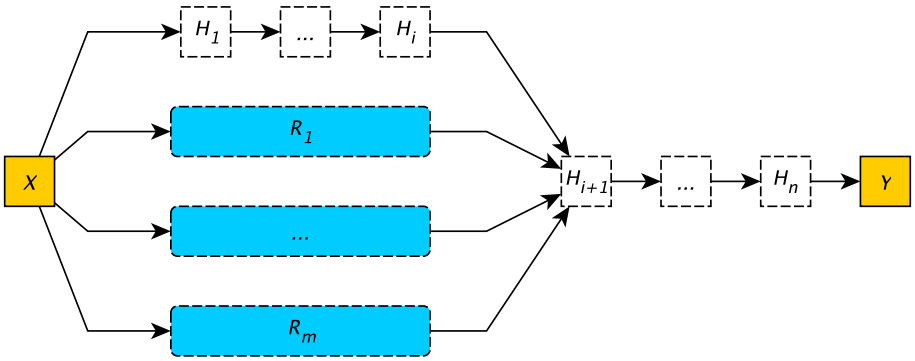


Fig. 1: An example of a network's architecture after the insertion of modules derived from logic formulæ.

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 with 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, *(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 for-

mulæ 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 pointing out 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 $H_1, \ldots, H_n$ 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 built 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. Any other synapsis – there including all hidden synapses among layers $H_1, \ldots, H_i$, 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. It should be noted 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 publicly-available implementation of the algorithm is available as part of the PSyKI framework [24].

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 $\psi_1, \psi_2, \dots$ constituting 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:

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

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:

$$\begin{aligned} \mathit{class}(\bar{X}, \mathtt{y}_1) &\leftarrow p_1(\bar{X}) \wedge p_2(\bar{X}) \\ \mathit{class}(\bar{X}, \mathtt{y}_2) &\leftarrow p'_1(\bar{X}) \wedge p'_2(\bar{X}) \\ \mathit{class}(\bar{X}, \mathtt{y}_3) &\leftarrow p''_1(\bar{X}) \wedge p''_2(\bar{X}) \end{aligned}$$

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

3.2 Fuzzy Logic Formulæ as Neural Modules

Formula	C. interpretation	Formula	C. interpretation
$[\![\neg\phi]\!]$	$\eta(1 - [\![\phi]\!])$	$[\![\phi \leq \psi]\!]$	$\eta(1 + [\![\psi]\!] - [\![\phi]\!])$
$[\![\phi \wedge \psi]\!]$	$\eta(min([\![\phi]\!], [\![\psi]\!]))$	$[\![class(\bar{X}, \mathtt{y}_i) \leftarrow \psi]\!]$	$[\![\psi]\!]^*$
$[\![\phi \vee \psi]\!]$	$\eta(max([\![\phi]\!], [\![\psi]\!]))$	$[\![\text{expr}(\bar{X})]\!]$	$\text{expr}([\![\bar{X}]\!])$
$[\![\phi = \psi]\!]$	$\eta([\![\neg(\phi \neq \psi)]\!])$	$[\![\mathtt{true}]\!]$	1
$[\![\phi \neq \psi]\!]$	$\eta(\lvert[\![\phi]\!] - [\![\psi]\!]\rvert)$	$[\![\mathtt{false}]\!]$	0
$[\![\phi > \psi]\!]$	$\eta(\frac{1}{2} + [\![\phi]\!] - [\![\psi]\!])$	$[\![X]\!]$	x
$[\![\phi \geq \psi]\!]$	$\eta(1 + [\![\phi]\!] - [\![\psi]\!])$	$[\![\mathtt{k}]\!]$	k
$[\![\phi < \psi]\!]$	$\eta(\frac{1}{2} + [\![\psi]\!] - [\![\phi]\!])$	$[\![p(\bar{X})]\!]^{**}$	$[\![\psi_1 \vee \ldots \vee \psi_k]\!]$

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

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

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 is $\bar{X}$ a tuple of logic variables. Similarly, $\mathtt{k}$ is a numeric constant, and k is the corresponding real value, whereas $\mathtt{k}_i$ 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 $\bar{X}$.

Before 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 [19] reported in Table 1.

Accordingly, we encode each formula via $[\![\cdot]\!]$ 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. Scalars are then clipped into the $[0, 1]$ range, via function $\eta : \mathbb{R} \to [0, 1]$ defined as follows:

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

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 – a rule that always correctly classifies w.r.t. the ground truth – for class *flush*, i.e., all cards have the same suit, as follow:

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

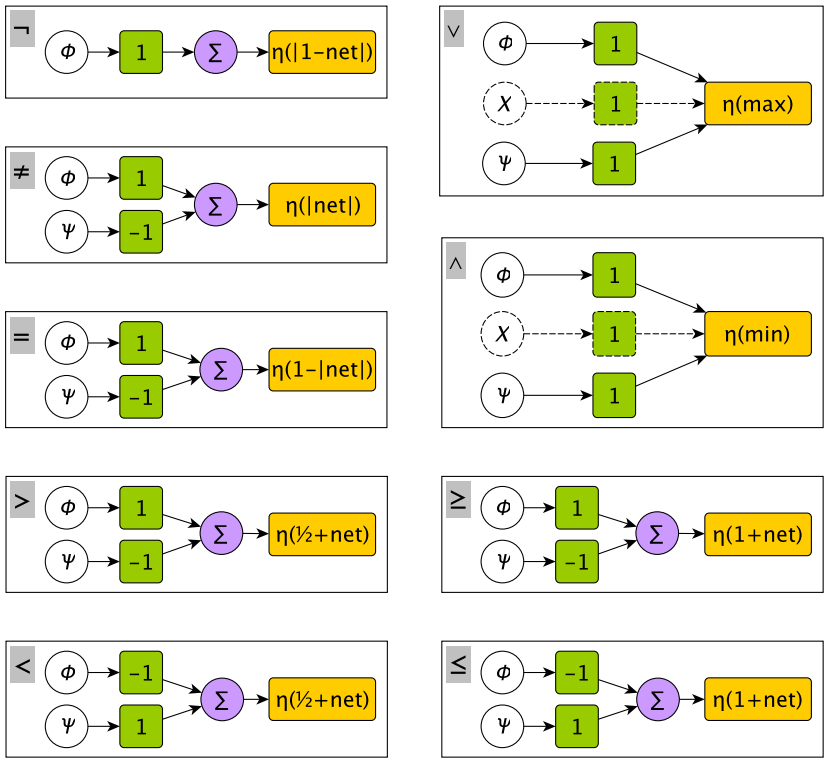
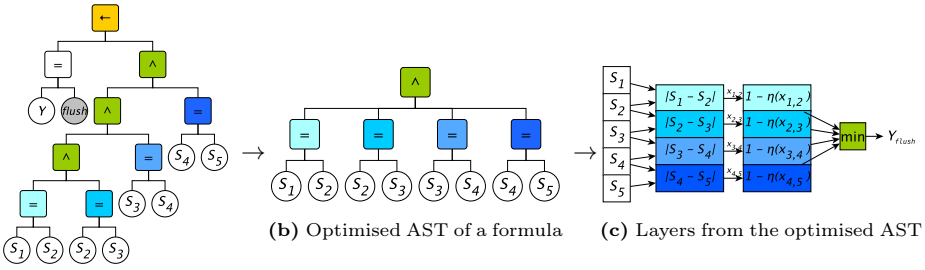


Fig. 2: Mapping of formulæ into neurons. White circles are input variables (I), green boxes represent the corresponding weights (W), purple circles are the sum of the weighted inputs ($W \times I$). Yellow rectangles are activation functions, net is the output of $W \times I$, max and min respectively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



(a) AST of a formula

(b) Optimised AST of a formula

(c) Layers from the optimised AST

Fig. 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.

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

It is worth noting that there could be an infinite number of different – yet idempotent – logic programs. This means that the same logic rule can be written in multiple different ways without changing the final result. However the corresponding mapping into neurons would be different. For the scope of this paper, we do not investigate how different but idempotent logic rules affect the behaviour of the resulting NN. We leave this topic for a future work.

3.3 Comparison with the literature

The main difference between KINS and the majority of existing SKI algorithms is the sort of knowledge supported. Whereas propositional logic is the most common kind of logic formalism, we rely on stratified Datalog with negation. This provides the user of the algorithm with more choices in the selection of the knowledge to inject; also, the knowledge itself can be easily written in an intensionally way. Moreover, we define a mapping not only for the main logic operators – *conjunction*, *disjunction* and *negation* – but also for arithmetic comparison operators. KINS can handle constants of real numbers, therefore it can be used with datasets that have continuous features with no need for discretisation. On the other hand, KBANN supports propositional logic and is limited to categorical data types.

Another difference with KBANN and other SKI algorithms is the lack of hyper-parameters to fine-tune except for the index of the layer where to actually perform the injection (e.g., $i+1$ for the hidden layer H_{i+1} in Figure 1). Usually, these parameters are real numbers, and their interpretation it is not trivial. This results in expensive grid-search approaches to find the optimal values. For instance, in the case of KBANN there are two possible parameters to fine-tune, namely, ω and λ. The first one affects the activation functions and the weights of the NN: the author proposes a default value of $\omega = 4$ (that we maintain for the experiments) by empirical tries. The second one is a regularisation term of the cost factor for the loss function.

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/pikalab-unibo/jlc-experiments-2022. Python implementations of KINS and KBANN are publicly available on PSyKI [24].

The design of the experiments is as follows. We choose two different datasets – namely the primate splice-junction gene sequences (PSJGS) dataset and the Wisconsin breast cancer dataset (WBC) [13] – and we then train neural networks on them, with and without knowledge injection via KINS. We also run additional experiments using KBANN – a well known SKI procedure from the literature, cf. Section 2 –, other than non-neural predictors—namely decision tree classifiers (CART [7]) and K nearest neighbours [10] with $K = 3$ and $K = 5$. The Student-T test is then exploited to assess the statistical significance of the performance improvements attained by NN undergoing injection via SKI.

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 [13]. 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: $X_{-30}, \ldots, X_{-1}, X_{+1}, \ldots, X_{+30}$. There, variable $X_{\pm i}$ denotes the value of the nucleotide in position $\pm i$, which is represented via ad-hoc constants (namely, $\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}$). For the sake of readability, we write $\bar{X}$ in place of the full sequence of variables $X_{-30}, \ldots, X_{+30}$.

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 4a 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:

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

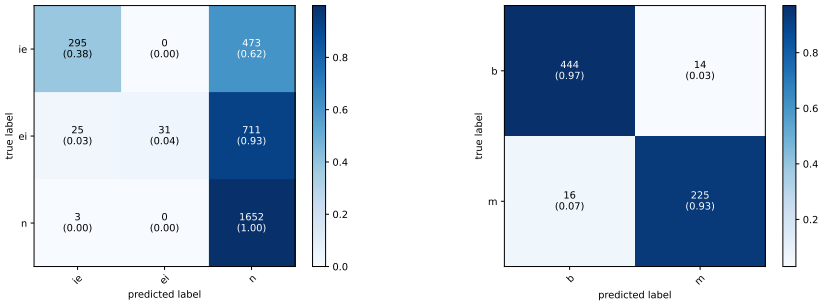
While this is far-from-perfect knowledge describing the entire domain with no or few errors, it is still good enough to positively affect the training of the predictor.

Class	Logic Formulation
EI	$class(\bar{X}, \texttt{ei}) \leftarrow X_{-3} = \texttt{m} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g} \wedge X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{r} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{t} \wedge \neg(ei\_stop(\bar{X}))$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{g} \wedge X_{-1} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{g} \wedge X_{-2} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{g} \wedge X_{-3} = \texttt{a}$
IE	$class(\bar{X}, \texttt{ie}) \leftarrow pyramidine\_rich(\bar{X}) \wedge \neg(ie\_stop(\bar{X})) \wedge X_{-3} = \texttt{y} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g}$
	$pyramidine\_rich(\bar{X}) \leftarrow 6 \leq (X_{-15} = \texttt{y} + \ldots + X_{-6} = \texttt{y})$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{g} \wedge X_{+4} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{g} \wedge X_{+5} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{a}$

Table 2: Datalog formulæ describing DNA classification criteria generated from the original one.

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

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.



(a) Confusion matrix of the PSJGS classification rules.

(b) Confusion matrix of the WBC classification rules.

Fig. 4: Confusion matrices of the rules used as prior knowledge in the experiments.

Methodology To make our experiments comparable with already existing literature benchmarks, we follow the very same method used by Towell and Shavlik in [29] with few changes. We use 10-fold cross validation using all the 3190 available records of the dataset instead of just 1000 randomly-chosen records like in [29]. Since the experiments depend on how the records are split into folders, we repeat the overall experiment 30 times – instead of just 10 of the original experiments – to improve the significance of the results. We also use three rules – the original ones for classes `ei` and `ie`, plus the fictional rule described in Equation (4) for the third class. The original experiment considers records of class n if both the outputs of the two rules for `ei` and `ie` are lower than the threshold of 0.5. Before training, we preprocess the dataset to obtain new features from the original ones by performing hot encoding without altering the information of the dataset. Every value of the DNA sequence is encoded into a 4 value binary array ($\texttt{a} \to [1,0,0,0]$, $\texttt{c} \to [0,1,0,0]$, $\texttt{g} \to [0,0,1,0]$, $\texttt{t} \to [0,0,0,1]$, $\texttt{d} \to [1,0,1,1]$, and so on).

More precisely, each time we train an instance of KINS we leverage on a NN structured as follow:

- input layer, 240 neurons ($60 \cdot 4$);
- hidden fully connected layer, 40 neurons with rectified linear unit as activation function;
- output fully connected layer, 3 neurons with softmax as activation function.

The optimiser used for training is Adam [21], categorical cross-entropy as loss function, and a batch size of 32. We use the same stopping criteria used in [29], 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. This network architecture empirically performs very well for PSJGS task given the method for validation used.

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

4.2 Wisconsin Breast Cancer

We validate the algorithm on another well-known benchmark: the Wisconsin breast cancer dataset (WBC) [13]. The dataset comes in three variants: *(i)* Wisconsin breast cancer (WBC) is the original dataset that we use for experiments; *(ii)* Wisconsin diagnostic breast cancer (WDBC) is a later version of WBC that has features of real numbers; *(iii)* Wisconsin prognostic breast cancer (WPBC) is a later and smaller dataset that concerns breast cancer prognoses instead of diagnoses. WBC consists of 699 records, each of them contains 9 categorical features – `ClumpThickness`, `UniformityCellSize`, `UniformityCellShape`, `MarginalAdhesion`, `SingleEpithelialCellSize`, `BareNuclei`, `BlandChromatin`, `NormalNucleoli`, `Mitoses` – encoded with an integer value from 1 to 10. There exist few missing values (16) originally reported with a question mark symbol that we encode with the value of 0. One feature (`Mitoses`) has only 9 different values. One record can be classified as benign or malignant, respectively split into 458 (65.5%) and 241 (34.5%) records.

The dataset has been introduced in the early '90s in [5, 25, 31, 32]; new records have been added during years until it has reached the current size of 699 records. This dataset does not come with classification rules like the PSJGS dataset. Therefore we provide a set of propositional classification rules that is elicited in Table 4. Rules have been generated using the CART [7] decision tree algorithm (with default scikit-learn [27] hyper-parameters). The overall accuracy of the rules on the whole dataset of almost 95.57%. Figure 4b reports the confusion matrix of the rules for the WBC task.

Methodology Like for the PSJGS task, we use the same methodology to conduct experiments on this dataset: we use 10-fold cross validation to validate the performance of the algorithms using all the 699 available records (the same criterion has been applied in [17, 28]). We stay strict to the three early stopping condition mentioned in Section 4.1 with the only exception of raising from 90% to 95% the threshold for the third condition since the WBC task is simpler than the PSJGS task. We preprocess the dataset by binarising each feature since KBANN does not directly support integers in the injectable rules. The NN upon with KINS relies is structured as follow:

- input layer, 90 neurons ($9 \cdot 10$);
- hidden fully connected layer, 20 neurons with rectified linear unit as activation function;
- output fully connected layer, 2 neurons with softmax as activation function.

Class	Logic Formulation
benign	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $\wedge\ UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1 \wedge$ $MarginalAdhesion \neq 5 \wedge NormalNucleoli = 7 \wedge$ $BlandChromatin = 4$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1 \wedge$ $MarginalAdhesion = 5 \wedge BlandChromatin = 3$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape = 2$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei = 1 \wedge$ $Mitoses \neq 3 \wedge MarginalAdhesion \neq 10$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize = 1 \wedge NormalNucleoli \neq 4 \wedge$ $BareNuclei \neq 5$
	$class(\bar{X}, \texttt{benign}) \leftarrow UniformityCellSize = 1 \wedge NormalNucleoli \neq 4 \wedge$ $BareNuclei = 5 \wedge ClumpThickness \neq 4$
malignant	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1 \wedge$ $MarginalAdhesion \neq 5 \wedge NormalNucleoli \neq 7$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1 \wedge$ $MarginalAdhesion \neq 5 \wedge NormalNucleoli = 7 \wedge$ $BlandChromatin \neq 4$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei \neq 1 \wedge$ $UniformityCellShape \neq 2 \wedge UniformityCellShape \neq 1 \wedge$ $MarginalAdhesion = 5 \wedge BlandChromatin \neq 3$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei = 1 \wedge$ $Mitoses \neq 3 \wedge MarginalAdhesion = 10$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge BareNuclei = 1 \wedge$ $Mitoses = 3$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize \neq 1 \wedge NormalNucleoli \neq 4 \wedge$ $BareNuclei = 5 \wedge ClumpThickness = 4$
	$class(\bar{X}, \texttt{malignant}) \leftarrow UniformityCellSize = 1 \wedge NormalNucleoli = 4$

Table 4: Datalog formulæ describing breast cancer classification criteria.

Optimiser, loss function, and batch size are the same used in Section 4.1. Rules are reported in Table 4 and they are considered all mutable for the experiments, i.e. trainable. This architecture allows the network to achieve empirically-good performance for the WBC task.

5 Discussion

5.1 Primate Splice-Junction Gene Sequences

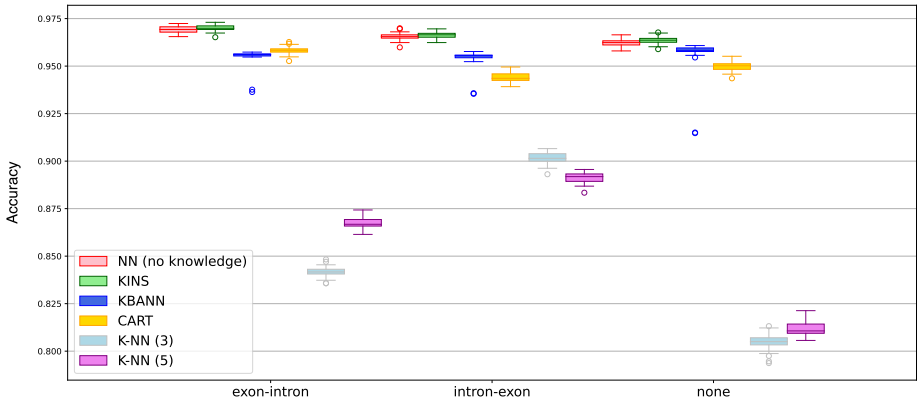


Fig. 5: PSJGS per class accuracy distributions over 30 experiments. In order from left to right: base neural network with no knowledge (red), KINS (green), KBANN (blue), CART decision tree (yellow), k-nearest neighbours with 3 (silver) and with 5 (violet) neighbours.

We validate KINS by injecting prior knowledge from Table 2 in different layers of the NN previously described in Section 4.1. Best results are obtained when injection is immediately performed at the first layer. Following the aforementioned methodology, we have 10 predictors trained on 2,871 records each – the remaining 319 records are used for validation – for each experiment. Because the training is affected by the splitting of the records, we run 30 experiments with random weights initialisation to increase the statistical relevance.

After the injection of the prior knowledge into the first layer and training, the mean overall accuracy of the 30 experiment on the validation set is 95%. Single mean class accuracies are: (`ei`) 97%, (`ie`) 96.63%, (`n`) 96.36%.

We execute 30 additional runs using the same base architecture NN *without* the injection of any knowledge obtaining the following results: (mean accuracy) 94.85%, (`ei`) 96.92%, (`ie`) 96.55%, (`n`) 96.21%. After computing Student's T-test on the two distributions we reject the null hypothesis (p-value = 0.015): predictors generated from KINS have better accuracies with statistic relevance.

Predictor	accuracy	class acc.	precision	recall	f1-score	class
KINS	0.95	0.97	0.932	0.945	0.938	exon − intron
		0.966	0.927	0.935	0.93	intron − exon
		0.964	0.97	0.959	0.965	none
NN (no knowledge)	0.948	0.969	0.929	0.945	0.937	exon − intron
		0.965	0.923	0.936	0.929	intron − exon
		0.962	0.971	0.956	0.963	none
KBANN	0.932	0.955	0.891	0.919	0.904	exon − intron
		0.954	0.912	0.889	0.9	intron − exon
		0.956	0.959	0.958	0.958	none
CART	0.926	0.958	0.907	0.922	0.914	exon − intron
		0.944	0.878	0.893	0.885	intron − exon
		0.95	0.96	0.943	0.951	none
K-NN (3)	0.774	0.842	0.615	0.925	0.738	exon − intron
		0.901	0.745	0.901	0.815	intron − exon
		0.805	0.968	0.645	0.774	none
K-NN (5)	0.785	0.868	0.66	0.935	0.773	exon − intron
		0.891	0.713	0.925	0.804	intron − exon
		0.812	0.979	0.652	0.782	none

Table 5: Average-value metrics for the PSJGS task computed on 30 runs for each predictor. It is worth mentioning that the accuracies obtained by KINS are statistically better than the ones obtained by NN without knowledge with a p-value = 0.015.

The improvement of the accuracy using our injection method is significant even with imperfect knowledge.

We also run additional experiments using KBANN – with the same knowledge in Table 2 and with the hyper-parameter $\omega = 4$ – and using predictors that are not NN with the same setup: a decision tree classifier – CART [7] – and k-nearest neighbours [10] with 3 and 5 neighbours. Results of all algorithms are reported in Table 5 and in Figure 5. Connectionist solutions – the base network, KINS, KBANN – have better performance than the other algorithms. CART performs well, with an overall mean accuracy closes to KBANN (0.926 and 0.932 respectively). Both k-nearest neighbours algorithms are clearly inferior in classifying DNA sequences for the PSJGS task.

5.2 Wisconsin Breast Cancer

We validate KINS by injecting prior knowledge from Table 4 in different layers of the NN previously described in 4.2. Like for the WBC experiments, best results are obtained when the injection is performed at the first hidden layer. Following the aforementioned methodology, we have 10 predictors trained on 629 records each – the remaining 70 records are used for validation – for each experiment. We run 30 experiments with random weights initialisation to increase the statistical relevance.

After the injection of knowledge into the first layer and training, the mean overall accuracy of the 30 experiment on the validation set is 96.7%.

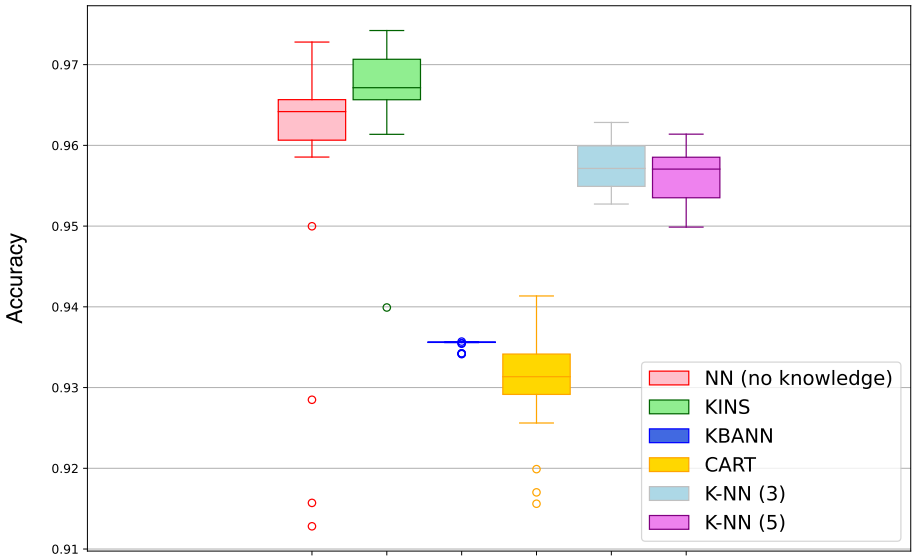


Fig. 6: WBC accuracy distributions over 30 experiments. From left to right: base neural network with no knowledge (red), KINS (green), KBANN (blue), CART decision tree (yellow), k-nearest neighbours with 3 (silver) and with 5 (violet) neighbours.

We execute 30 additional runs using the same base architecture NN *without* the injection of any knowledge obtaining a mean accuracy value of 96%. After computing Student's T-test on the two distributions we reject the null hypothesis (p-value = 0.01): predictors generated from KINS have better accuracies with statistic relevance.

We include in our analysis additional experiments using KBANN – with the same knowledge in Table 2 and with the hyper-parameter $\omega = 1$ (empirically a good value for this task) – and using predictors that are not NN with the same setup: a decision tree classifier (CART) and k-nearest neighbours with 3 and 5 neighbours. Results of all algorithms are reported in Table 6 and in Figure 6. The base network and KINS show extremely high performance—comparable or even better than the ones obtained by connectionist solutions reported in [17]. They are followed by the K-nearest neighbours, while KBANN and CART have considerably worse performance. We speculate that the main reason for KBANN to perform that way is due to the prior knowledge the algorithm is provided with. Since knowledge is already quite good KBANN cannot exploit it as much as in the PSJGS task; moreover, the rules are quite straightforward different from the PSJGS knowledge where predicates are more articulated.

Predictor	accuracy	precision	recall	f1-score	class
KINS	0.967	0.978	0.972	0.975	benign
		0.949	0.958	0.952	malignant
NN (no knowledge)	0.96	0.976	0.963	0.967	benign
		0.937	0.954	0.944	malignant
KBANN	0.935	0.959	0.943	0.95	benign
		0.898	0.921	0.908	malignant
CART	0.931	0.944	0.952	0.948	benign
		0.91	0.891	0.899	malignant
K-NN (3)	0.957	0.962	0.974	0.968	benign
		0.951	0.925	0.937	malignant
K-NN (5)	0.956	0.96	0.975	0.967	benign
		0.952	0.921	0.935	malignant

Table 6: Average-value metrics for the WBC task computed on 30 runs for each predictor. It is worth mentioning that the accuracies obtained by KINS are statistically better than the ones obtained by NN without knowledge with a p-value = 0.01.

6 Conclusion

In this work we discuss KINS as 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. Experiments on both PSJGS and WBC datasets show that KINS is statistically relevant in improving the final accuracy of the original predictor, even when the performance are already very good and with incorrect knowledge. The selected task has some of the common criticalities of ML classification tasks, in particular data set size limitation and unbalanced classes. Moreover, prior knowledge provided is far from perfect. Results show that our approach can improve network 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).

Bibliography

[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

[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), URL 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, pp. 19–22, CEUR-WS.org (2008), URL 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), URL http://www.aaai.org/Library/AAAI/1986/aaai86-033.php

[5] Bennett, K.P., Mangasarian, O.L.: Robust linear programming discrimination of two linearly inseparable sets. Optimization Methods and Software **1**(1), 23–34 (1992), https://doi.org/10.1080/10556789208805504

[6] Besold, T.R., d'Avila Garcez, A.S., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kühnberger, K., Lamb, L.C., Lima, P.M.V., de Penning, L., Pinkas, G., Poon, H., Zaverucha, G.: Neural-symbolic learning and reasoning: A survey and interpretation. In: Hitzler, P., Sarker, M.K. (eds.) Neuro-Symbolic Artificial Intelligence: The State of the Art, Frontiers in Artificial Intelligence and Applications, vol. 342, pp. 1–51, IOS Press (2021), https://doi.org/10.3233/FAIA210348

[7] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. Routledge (1984), https://doi.org/10.1201/9781315139470

[8] 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

[9] 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

[10] Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory **13**(1), 21–27 (1967), https://doi.org/10.1109/TIT.1967.1053964
[11] 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
[12] 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
[13] Dua, D., Graff, C.: UCI machine learning repository (2017), URL http://archive.ics.uci.edu/ml
[14] 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
[15] d'Avila Garcez, A.S., Gabbay, D.M.: Fibring neural networks. In: McGuinness, D.L., Ferguson, G. (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, pp. 342–347, AAAI Press / The MIT Press (2004), URL http://www.aaai.org/Library/AAAI/2004/aaai04-055.php
[16] 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
[17] Ghosh, S., Mondal, S., Ghosh, B.: A comparative study of breast cancer detection based on SVM and MLP BPN classifier. In: 2014 First International Conference on Automation, Control, Energy and Systems (ACES), pp. 1–4, IEEE (2014), https://doi.org/10.1109/ACES.2014.6808002
[18] 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
[19] Hay, L.S.: Axiomatization of the infinite-valued predicate calculus. The Journal of Symbolic Logic **28**(1), 77–86 (1963), ISSN 00224812, URL http://www.jstor.org/stable/2271339
[20] 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, pp. 2410–2420, The Association for Computer Linguistics (2016), https://doi.org/10.18653/v1/p16-1228
[21] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Repre-

sentations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), URL http://arxiv.org/abs/1412.6980

[22] Lipton, Z.C.: The mythos of model interpretability. Communications of the ACM **61**(10), 36–43 (2018), https://doi.org/10.1145/3233231

[23] Magnini, M., Ciatto, G., Omicini, A.: KINS: Knowledge injection via network structuring. In: Calegari, R., Ciatto, G., Omicini, A. (eds.) CILC 2022 – Italian Conference on Computational Logic, CEUR Workshop Proceedings, vol. 3204, pp. 254–267, CEUR-WS (2022), ISSN 1613-0073, URL http://ceur-ws.org/Vol-3204/paper_25.pdf

[24] 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, Lecture Notes in Computer Science, vol. 13283, chap. 6, pp. 90–108, Springer (2022), ISBN 978-3-031-15564-2, https://doi.org/10.1007/978-3-031-15565-9_6, 4th International Workshop, EXTRAAMAS 2022, Virtual Event, May 9–10, 2022, Revised Selected Papers

[25] Mangasarian, O.L., Wolberg, W.H.: Cancer diagnosis via linear programming. Computer Sciences Technical Report 958, Computer Sciences Department, University of Wisconsin – Madison (Aug 1990), URL https://minds.wisconsin.edu/bitstream/handle/1793/59346/TR958.pdf

[26] 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

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011), URL https://www.jmlr.org/papers/v12/pedregosa11a.html

[28] Salama, G.I., Abdelhalim, M., Zeid, M.A.e.: Breast cancer diagnosis on three different datasets using multi-classifiers. International Journal of Computer and Information Technology (IJCIT) **1**(1), 36–43 (2012), ISSN 2279-0764, URL https://www.ijcit.com/archives/volume1/issue1/Paper010105.pdf

[29] Towell, G.G., Shavlik, J.W.: Knowledge-based artificial neural networks. Artificial Intelligence **70**(1-2), 119–165 (1994), https://doi.org/10.1016/0004-3702(94)90105-8

[30] 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), URL http://papers.nips.cc/paper/638-network-structuring-and-training-using-rule-based-knowledge

[31] Wolberg, W., Mangasarian, O., Coleman, T., Li, Y.: Pattern recognition via linear programming: Theory and application to medical diagnosis. Com-

puter Sciences Technical Report 878, Computer Sciences Department, University of Wisconsin – Madison (1989), URL https://minds.wisconsin.edu/bitstream/handle/1793/59186/TR878.pdf

[32] Wolberg, W.H., Mangasarian, O.L.: Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the National Academy of Sciences **87**(23), 9193–9196 (Dec 1990), https://doi.org/10.1073/pnas.87.23.919

[33] 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), URL https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html

[34] 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), URL http://proceedings.mlr.press/v80/xu18h.html