EXPECTATION

PERSONALIZED EXPLAINABLE ARTIFICIAL INTELLIGENCE FOR DECENTRALIZED AGENTS WITH HETEROGENEOUS KNOWLEDGE

D2.1
Tech report on symbolic knowledge extraction and injection

[M12]

Version: 1.0
Last Update: 23/03/2022
Distribution Level: CO

Distribution levels*

Project supported by

PU = Public;
RE = Restricted to a group within the given consortium;
PP = Restricted to other program participants (including commission services);
CO = Confidential, only for the members of the EXPECTATION Consortium (including commission services);

Project supported by

EXPECTATION

The EXPECTATION Project Consortium groups Organizations involved:

Partner Name	**Short name**	**Country**
University of Bologna / Dep. Of Computer Science and Engineering	UNIBO	Italy

Document Identity

Creation Date:	05/10/2021
Last Update:	23/03/2022

Revision History

Version	**Edition**	**Author(s)**	**Date**
1	1	All PIs	23/03/2022
Comments:			

Project supported by

Symbolic Knowledge Extraction and Injection from/into Sub-symbolic Predictors

Technical Report

Federico Sabbatini[1*] Andrea Agiollo[2*] Giovanni Ciatto[2]
Andrea Omicini[2]

[*] Co-first authors

[1] Dipartimento di Scienze Pure e Applicate (DiSPeA)
Università degli Studi di Urbino Carlo Bo, Italy
f.sabbatini@unibo.it, f.sabbatini1@campus.uniurb.it

[2] Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna, Italy
{andrea.agiollo, giovanni.ciatto, andrea.omicini}@unibo.it

In this paper we focus on the problem of working around the opacity issue of sub-symbolic machine-learning predictors—such as neural networks, support vector machines, decision trees, random forests, or linear models. We do so by promoting two complementary activities, namely *symbolic knowledge extraction* (SKE) and *injection* (SKI) from and into sub-symbolic predictors. There, we consider as symbolic any intelligible language which is naturally interpretable for both human beings and computers. Along this line, in this work we propose a general meta-model for both SKE and SKI, as well as two taxonomies by which SKE and SKI methods can be classified. We do so under an explainable AI perspective, hence highlighting how both SKE and SKI methods can be exploited to mitigate or workaround the aforementioned opacity issue. Notably, our taxonomies are attained by surveying and classifying the existing SKE/SKI methods from the literature, and by generalising the results of previous surveys targeting particular sub-topics of either SKE or SKI alone. More precisely, we analyse roughly 90 methods for SKE and roughly 70 methods for SKI, and categorise them according to their purpose, functioning, expected input/output data and predictor types. These meta-models let us define practical guidelines for data scientists willing to select the most adequate SKE/SKI method for their needs, as well as suggestions for researchers interested in filling the gaps of the current state of the art, or developers willing to implement SKE or SKI software technologies.

Contents

1 Introduction

In the context of artificial intelligence (AI), more and more critical applications are being developed by relying on machine learning (ML). This subtends a data-driven approach to the engineering of intelligent computational systems, where hard-to-code tasks are semi-automatically learned from data rather than manually programmed by human developers. Examples of tasks which can be learned this way range from text to speech or image recognition, stepping through time series forecasting, clustering, and so on. Applications are manifold, and make our life easier in many ways—e.g., via speech-to-text applications, email spam and malware filtering, customer profiling, automatic translation, virtual personal assistants, and so forth.

Learning, in particular, is automated via ML algorithms, often implying *numeric* processing of data—which in turn enables the detection of fuzzy patterns or statistically-relevant regularities in the data, that algorithms can learn to recognize. This is fundamental to support the automatic acquisition of otherwise hard-to-formalise behaviours for computational systems. However, such flexibility comes at the cost of poorly *interpretable* solutions, as state-of-the-art *sub-symbolic* predictors – such as neural networks – are often exploited behind the scenes.

By "interpretable" we here mean that the expert human user may contemplate the computational system and understand its behaviour—which is therefore predictable. This property is not always required. However, there exist safety-, value-, or ethic-critical applications where humans must be in full control of the computational systems supporting their decisions or aiding their actions. In those cases, the lack of interpretability is a no-go. Consider for instance the case of automatic stock trading, diagnosis support, or self-driving car systems: would anyone trust these kinds of applications if not even the engineers realising them can fully understand and predict – and therefore control – their behaviour?

State-of-the-art ML systems rely on a bunch of well-established data mining predictors, such as neural networks, support vector machines, decision trees, random forests, or linear models. Despite the latter sorts of predictors being often considered interpretable in the general case, as the complexity of the problem at hand increases (e.g., dimensionality of the available data) trained predictors become more complex, hence harder to contemplate, and therefore less interpretable. Nevertheless, these mechanisms have penetrated the modern practices of data scientists because of their flexibility, and expected effectiveness—in terms of predictive performance. Unfortunately, a number of experts have empirically observed an inverse proportionality relation among interpretability and predictive performance [29, 146]. This is the reason why data-driven engineering efforts starting today and targeting critical application scenarios will eventually have to choose whether to prioritise predictive performance or interpretability. We call it the interpretability–performance trade-off.

In this paper, we focus on the problem of working around the interpretability–performance trade-off. We do so by promoting two complementary activities, namely *symbolic knowledge extraction* (SKE) and *injection* (SKI) from and into sub-symbolic predictors. In both cases, "symbolic" refers to the way knowledge is represented. In particular, we consider as symbolic any intelligible language which is naturally interpretable for both human beings and computers. This includes a number of logic formalisms, and excludes the fixed-sized tensors of numbers commonly exploited in sub-symbolic ML.

Broadly speaking, SKE is the process of distilling the knowledge a sub-symbolic predictor has grasped from data into symbolic form. This can be exploited to provide *post-hoc* explanations for otherwise poorly interpretable sub-symbolic predictors. More generally, SKE enables the *inspection* of the sub-symbolic predictors it is applied to, making it possible for the human designer to figure out how exactly they will behave. This, in turn, allows data scientists to *debug* sub-symbolic predictors, similarly to what computer scientists would do with misbehaving programs.

Conversely, SKI is the inverse process of letting a sub-symbolic predictor follow the symbolic knowledge possibly encoded by its human designers. This can come in the form of *constraint* – aimed at preventing the predictor from learning something the designers want to avoid –, or *suggestion* – aimed at encouraging the predictor to learn something the designers consider as positive –, when not both. More generally, SKI enables a higher degree of *control* over a sub-symbolic predictor and its behaviour, constraining it with human-like common-sense—suitably encoded into symbolic form. In this way, data scientists may *correct* a misbehaving predictor or train one despite the shortage of examples describing a particular phenomenon. For example, symbolic knowledge may encode impossible or meaningless situations to be avoided (e.g., "if X has two legs, then X cannot be a cat"), as well as obvious (for humans) relations which may be hard (for predictors) to learn from data alone (e.g., "if X is Y's parent, then Y is X's child").

Furthermore, the greatest potential comes from the *combined* exploitation of SKE and SKI. Indeed, they can be exploited in cascade-like fashion where sub-symbolic predictors are *(i)* inspected (via SKE), *(ii)* debugged, and *(iii)* fixed (via SKI) by data scientists, as part of their ML workflows.

Along this line, we survey the literature with the purposes of *(i)* collecting and categorising the existing methods for SKE and SKI, and *(ii)* providing an overview of the state of the art. In particular, we analyse roughly 90 methods for SKE and roughly 70 methods for SKI, and categorise them according to their purpose, functioning, expected input/output data and predictor types. In doing so, we elicit a meta-model for SKE (resp. SKI) according to which existing and future extraction (resp. injection) methods can be categorised and described. These meta-models let us define practical guidelines for data scientists willing to select the most adequate SKE/SKI method for their need, as well as suggestions for researchers interested in filling the gaps of the current state of the art, or developers willing to implement SKE or SKI software technologies.

2 Background

2.1 Machine Learning Overview

A famous definition of machine learning from [125] states:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E

This definition is very loose, as it does not specify *(i)* what are the possible tasks, *(ii)* how performance is measured in practice, *(iii)* how / when experience should be provided to tasks, *(iv)* how exactly the program is supposed learn, and *(v)* under which form learnt information are represented. Accordingly, depending on the particular ways these aspects are tackled, a categorisation of the approaches and techniques for letting software agents learn may be drawn.

Three major approaches to ML exist, namely *supervised*, *unsupervised*, and *reinforcement* learning. Each approach is tailored on a well-defined pool of tasks, which may, in turn, be applied in a wide range of use case scenarios. Accordingly, differences among three major approaches can be understood by looking at the sorts of tasks T they support – commonly consisting of the estimation of some unknown relation –, and how experience E is provided to the learning algorithm.

In supervised learning, the learning task consists of finding a way to approximate an unknown relation given a sampling of its items—which constitute the experience. In unsupervised learning, the learning task consists of finding the best relation for a sample of items – which constitute the experience –, following a given optimality criterion intensionally describing the target relation. In reinforcement learning, the learning task consists of letting an agent estimate optimal plans given the reward it receives whenever it reaches particular goals. There, the rewards constitute the experience, while plans can be described as relations among the possible states of the world, the actions to be performed in those states, and the rewards the agents expects to receive from those actions.

Since several practical AI problems – such as image recognition, financial and medical decision support systems – can be reduced to *supervised* ML – which can be further grouped in terms of either *classification* or *regression* problems [189, 100] –, in the remainder of this subsection we focus on this set of ML problems.

2.1.1 Supervised Learning: Major Definitions

Within the scope of sub-symbolic supervised ML, a *learning algorithm* is commonly exploited to approximate the specific nature and shape of an unknown *prediction* function (or *predictor*) $\pi^* : \mathcal{X} \to \mathcal{Y}$, mapping data from an input space $\mathcal{X}$ into an output space $\mathcal{Y}$. There, common choices for both $\mathcal{X}$ and $\mathcal{Y}$ are, for instance, the set of vectors, matrices, or tensors of numbers of a given size—hence the sub-symbolic nature of the approach.

An important assumption significantly affecting both the theory and the practice of sub-symbolic supervised learning is that vectors, matrices, or tensors in $\mathcal{X}$ and $\mathcal{Y}$ are of *fixed* size—despite items in $\mathcal{X}$ may have different sizes than the items in $\mathcal{Y}$. Without loss of generality, in what follows we refer to items in $\mathcal{X}$ as n-dimensional vectors denoted as $\mathbf{x}$, whereas items in $\mathcal{Y}$ are m-dimensional vectors denoted as $\mathbf{y}$—despite matrices or tensors may be suitable choices as well.

To approximate function π^*, supervised learning assumes that a *learning algorithm* is in place. This algorithm computes the approximation by taking into account a number N

of *examples* of the form $(\mathbf{x}_i, \mathbf{y}_i)$ such that $\mathbf{x}_i \in X \subset \mathcal{X}$, $\mathbf{y}_i \in Y \subset \mathcal{Y}$, and $|X| \equiv |Y| \equiv N$. There, the set $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y\}$ is called *training* set, and it consists of $(n+m)$-dimensional vectors. The data set can be considered as the concatenation of two matrices, namely the $N \times n$ matrix of *input* data (X) and the $N \times m$ matrix of *expected output* data (Y). There, each $\mathbf{x}_i$ represents an instance of the input data for which the expected output value $\mathbf{y}_i \equiv \pi^*(\mathbf{x}_i)$ is known or has already been estimated. Notably, such sorts of ML problems are said to be "supervised" *because* the expected outputs Y are available. Furthermore, the function approximation task is called

regression if the components of Y consist of continuous or numerable – i.e., *infinite* – values,

classification if they consist of categorical – i.e., *finite* – values.

Many learning algorithms exist, and they work in quite different ways. However, the general layout of sub-symbolic supervised learning is the same in all cases. The learning algorithm assumes π^* to be a function from a given set of functions $\mathcal{H}$ called *hypothesis* space—i.e., $\pi^* \in \mathcal{H}$. In other words, the underlying assumption is that the unknown prediction function π^* exists, and it is of the form characterising all functions in $\mathcal{H}$. The algorithm performs an exploration of the *hypothesis* space $\mathcal{H}$ looking for the hypothesis function $\hat{\pi} \in \mathcal{H}$ that better fits the data in D—and that, therefore, better approximates π^*.

The goodness of the fitting among a hypothesis function $\hat{\pi}$ and the data can be assessed via an error function $\varepsilon : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ measuring the discrepancy among the expected outputs in Y and the values attained by applying $\hat{\pi}$ to X. For the sake of simplicity, we here consider the best hypothesis $\hat{\pi}$ the one item of $\mathcal{H}$ for which the total error is minimal, w.r.t. the data in D. Therefore, in theory, any sub-symbolic supervised learning process can be abstractly described via any of the following formulæ:

$$\hat{\pi} = \operatorname*{argmin}_{\pi \in \mathcal{H}} \left\{ \sum_{i=1}^{N} \varepsilon(\mathbf{y}_i,\ \pi(\mathbf{x}_i)) \right\}$$

Parameters and hyper-parameters Exploration of the hypothesis space is what is commonly referred to as "learning" or "training". Learning algorithms mostly differ for the strategy they follow to perform such exploration, other than the particular hypothesis spaces they support.

A common strategy followed by most learning algorithms leverages on the assumption that the hypothesis space is the set of all functions having the same shape, regulated by a given amount p of *parameters*—namely $\mathcal{H}_\Theta$ where $\Theta \subseteq \mathbb{R}^p$ is the space of parameters, enumerated by θ. Under such assumption, the formulation of supervised learning can be rewritten as the optimisation task of finding the optimal parameters θ^* among the ones in Θ:

$$\theta^* = \operatorname*{argmin}_{\theta \in \Theta} \left\{ \sum_{i=1}^{N} \varepsilon(\mathbf{y}_i,\ \pi_\theta(\mathbf{x}_i)) \right\}$$

where $\pi_\theta \in \mathcal{H}_\Theta$ is the particular function using the parameters in θ.

If all functions in $\mathcal{H}_\Theta$, as well as the error function ε, are differentiable w.r.t. θ, then the optimisation task can be tackled via gradient descent in the general case—despite better options may exist for particular shapes of the functions in $\mathcal{H}_\Theta$. Such need to rely on differentiable functions of vectors of real numbers is what forces many ML techniques into the sub-symbolic realm.

Notably, a hypothesis space is commonly generated by a particular assignment of a number q of *hyper-parameters* $\omega \in \mathbb{R}^q$. Each particular value of ω corresponds to a particular parameter space Θ, and therefore to a particular hypothesis space $\mathcal{H}_\Theta^\omega$. The hypothesis space may consist for instance of the set of all polynomials of a variables whose degree is b. That would imply the corresponding parameter space to comprehend all possible vectors having $\binom{b}{a}$ components. So, if $a = b = 1$, then $\mathcal{H}_\Theta^\omega$ is the set of all possible straight lines on a plane, i.e., polynomials parametrised by θ_1 and θ_2. For $a = 1$ and $b = 2$, $\mathcal{H}_\Theta^\omega$ corresponds to the set of all possible parabolas on a plane, i.e., polynomials parametrised by θ_1 and θ_2 and θ_3. A similar example may be built upon polynomials of 2 variables, and so on.

The difference among parameters and *hyper*-parameters is very important in practice. In fact, while parameters are *automatically* computed by the learning algorithm, hyper-parameters are not. They may be either guessed or estimated by trial-and-error by data scientists—hence representing a bottleneck in the automatisation of learning.

2.1.2 Overview on Learning Algorithms

Depending on the predictor family of choice, the nature of the admissible hypothesis spaces and learning algorithms may vary dramatically, as well as the predictive performance of the target predictor, and the whole efficiency of learning.

In the literature of machine learning, statistical learning, and data mining, a plethora of learning algorithms have been proposed along the years. Because of the "no free lunch" (NFL) theorem [205], however, no algorithm is guaranteed to outperform the others in all possible scenarios. For this reason, the literature and the practice of data science keeps leveraging on algorithms and methods whose first proposal was published decades ago. Most notable algorithms include for instance (deep) neural networks, decision trees, (generalised) linear models, nearest neighbours, support vector machines (SVM), random forests, and many others.

These algorithms can be categorised in several ways, for instance depending *(i)* on the supervised learning task they support (classification vs. regression), *(ii)* on when they consume data (lazy vs. eager), *(iii)* or on the underlying strategy adopted for learning (e.g., gradient descent, least squares optimisation), etc.

Some learning algorithms (e.g., neural networks) target regression problems, whereas others (e.g., SVM) target classification problems. Similarly, some target multi-dimensional outputs ($\mathbf{y} \in \mathbb{R}^m$, and $m > 1$), whereas others target mono-dimensional outputs ($m = 1$). Regressors are considered as the most general case, as other learning tasks can usually be defined in terms of mono-dimensional regression. Binary classifiers, for instance, can be treated as mono-dimensional regressors where admissible outputs lay in the interval $[0, 1]$, while multi-class (resp. multi-dimensional) classifiers (resp. regressors) can be treated as ensembles of multiple binary classifiers (resp. regressors).

The eager–lazy dichotomy relates to operational aspects of learning, and, in particular, to *when* training data is actually processed. This, in turn, affects the computational time required in the training and inference phases—i.e., when the predictor is exploited to draw predictions. In principle, the training phase of *lazy* predictors (e.g., nearest neighbours) is trivial and no data needs to be processed as training data is mostly processed in the inference phase. This makes the training phase quicker, at the expense of a slower inference phase. To mitigate this issue, in practice, indexing or grouping of training data may be exploited in the training phase, with the purpose of speeding up the inference phase. Conversely, *eager* predictors (e.g., linear models, neural networks, and virtually any other method mentioned so far) come with a full-fledged learning phase, where the unknown function binding inputs and outputs is approximated from training data. There, the learning phase carries the higher computational effort, and the inference phase is quick. In the remainder of this section, we focus on eager predictors as they actually produce an internal representation of data during their learning phase, which is a fundamental prerequisite for *decompositional* SKE methods (see section 4.1.1) as well as for SKI in general. Indeed, as further discussed in section 3, SKE requires some internal state from which knowledge can be extracted, whereas SKI requires some internal state to be altered in order to inject knowledge.

Finally, the learning strategy is inherently bound to the predictor family of choice. Neural networks, for instance, are trained via back-propagation [148] – a particular case of stochastic gradient descent (SDG, [204]), tailored on NN –, generalised linear models via Gauss' least squares method, decision trees via CART [25], etc. Despite all such algorithms may appear interchangeable in principle – because of the NFL theorem –, their malleability is very different in practice. For instance, the least squares method involves inverting matrices of order N – where N is the amount of available examples in the training set –, making the computational complexity of learning more than quadratic in time. Furthermore, in practice, convergence of the method is not guaranteed in the general case, while it is for generalised linear models—hence the reason why it is not adopted elsewhere. Thus, learning by least squares optimisation may become impractical for big data sets or for predictor families outside the scope of generalised linear models. Conversely, the SGD method involves arbitrarily-sized subsets of the data set (a.k.a. batches) to be processed a limited (i.e., controllable) amount of times. Hence, the complexity of SGD can be finely controlled and adapted to the computational resources at hand—e.g., by making the learning process incremental, and by avoiding all data to be loaded in memory. Furthermore, SGD can be applied to several sorts of predictor families (there including neural networks and generalised linear models), as it only requires the target function to be differentiable w.r.t. its parameters. For all these reasons, despite the lack of optimality guarantees, SGD is considered as very effective, scalable, and malleable in practice, hence why it is extensively exploited in the modern data science applications.

In the remainder of this subsection, we focus on two particular families of predictors – namely, decision trees and neural networks –, and their respective learning methods—

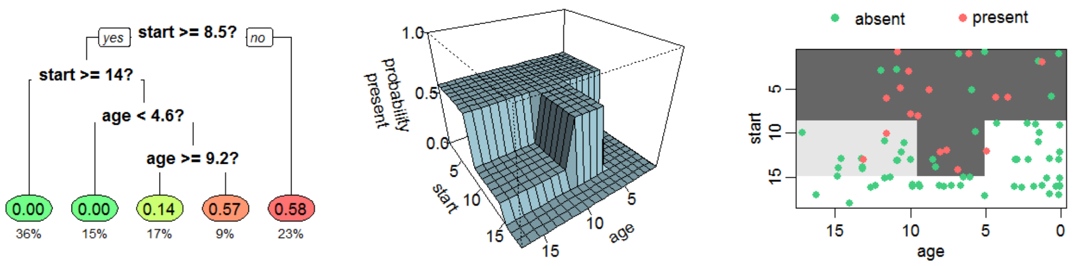

Figure 1: An example decision tree estimating the probability of kyphosis after spinal surgery, given the *age* of the patient and the vertebra at which surgery was *start*ed [203]. Notice that all decision trees subtend a partition of the input space, and that those trees themselves provide intelligible representations of *how* predictions are attained.

i.e., the CART [25] and back-propagation [148] algorithms. Notably, decision trees are relevant because of their user friendliness, whereas neural networks are relevant because of their predictive performance and flexibility.

Decision Trees Decision trees (DT) are particular sorts of predictors supporting both classification and regression tasks. In their learning phase, the input space is recursively *partitioned* through a number of splits (a.k.a. *decisions*) based on the input data X, in such a way that the prediction in each partition is constant, and the error w.r.t. the expected outputs Y is minimal, while keeping the total amount of partitions low as well. The whole procedure then synthesises a number of *hierarchical* decision rules to be followed whenever the prediction corresponding to any $x \in \mathcal{X}$ must be computed. In the inference phase, decision rules are orderly evaluated from the root to some leaf, to select the portion of the input space $\mathcal{X}$ containing x. As each leaf corresponds to a single portion of the input space, the whole procedure results in a single prediction for each x.

Differently from other families of predictors, the peculiarity of decision trees lays in the particular outcome of the learning process – namely, the *tree* of decision rules – which is naturally intelligible for humans and graphically representable in 2D charts. As further discussed in the remainder of this thesis, this property is of paramount importance whenever the inner operation of an automatic predictor must be interpreted by a human being.

Neural Networks Neural networks (NN) are biologically-inspired computational models, made of several elementary units (neurons) interconnected into a graph (commonly, *directed* and *acyclic*, a.k.a. DAG) via *weighted* synapses. Accordingly, the most relevant aspects of NN concern the inner functioning of neurons and the particular architecture of their interconnection.

Neurons are very simple numeric computational units. They accept n scalar inputs $(x_1, \ldots, x_n) = \mathbf{x} \in \mathbb{R}^n$ weighted by as many scalar weights $(w_1, \ldots, w_n) = \mathbf{w} \in \mathbb{R}^n$, and

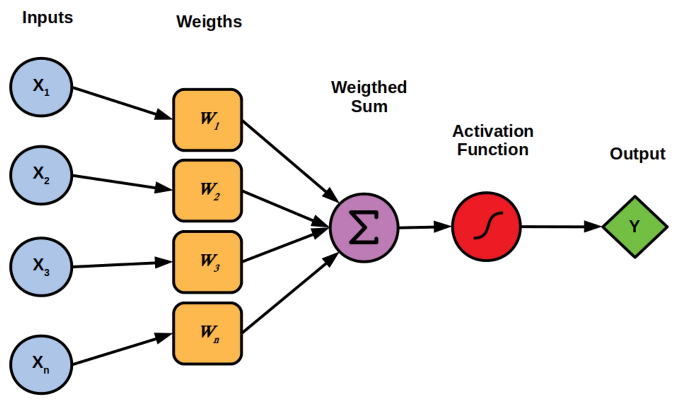

Figure 2: General structure of neural units in neural networks.

they process the linear combination $\mathbf{x} \cdot \mathbf{w}$ via an activation function [202] $\sigma : \mathbb{R} \mapsto \mathbb{R}$, producing a scalar output $y = \sigma(\mathbf{x} \cdot \mathbf{w})$, as depicted in Figure 2. The output of a neuron may become the input of many others, possibly forming *networks* of neurons having arbitrary topologies. These networks may be fed with any numeric information encoded as vectors of real numbers by simply letting a number of neurons produce constant outputs.

While virtually all topologies are admissible for NN, not all are convenient. A number of convenient *architectures* – roughly, patterns of well-studied topologies – have been proposed into the literature [193] to serve disparate purposes—far beyond the scope of supervised machine learning. Figure 3 overviews the current state of the art of NN architectures.

NN can be *trained* on numeric data via stochastic gradient descent and exploited into both supervised and unsupervised learning tasks such as classification, regression, and anomaly detection, depending on the particular architecture of choice. More precisely, while the training *automatically* sets up the weights of each neuron's ingoing synapses, the overall topology of the network is not allowed to vary. It is rather assumed to be *manually* engineered by data scientists.

Most common NN architectures are feed-forward, meaning that neurons are organised in successive *layers*, in such a way that neurons from layer i can only accept ingoing synapses from neurons of layers $j < i$. The first layer is considered the input layer, which is used to *feed* the whole network, while the last one is the output layer, where prediction are drawn. In these kind of architectures, inference lets information flow from the input to the output layers – assuming the weights of synapses are fixed –, while training lets information flow from the output to the input layers—provoking the variation of weights to minimise the prediction error of the overall network.

The recent success of deep learning [74] has proved the flexibility and the predictive performance of *deep* neural networks (DNN). 'Deep' here refers to the large amount of (possibly *convolutional*) layers. In other words, DNN can learn how to apply cascades of convolutional operations to the input data. Convolutions let the network spot relevant features into the input data, at possibly different scales. Hence why DNN are good at

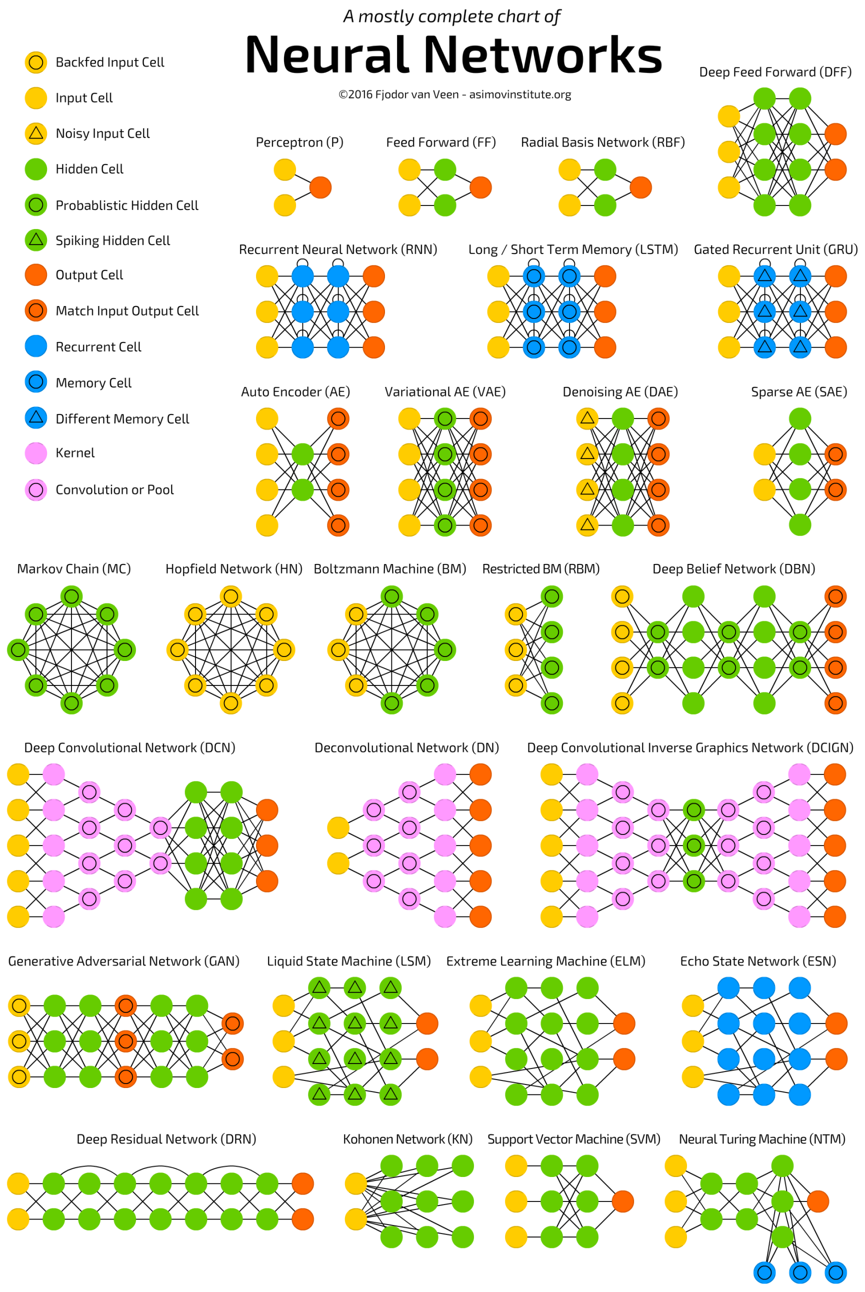
A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Figure 3: Admissible architectures for neural networks. [193]

solving complex pattern-recognition tasks, such as in computer vision or speech recognition. Unfortunately, however, unprecedented predictive performances of DNN come at the cost of their increased internal complexity and greater data greediness.

2.1.3 On the Nature of Sub-Symbolic Data

ML methods, and sub-symbolic approaches in general, represent data as (possibly multi-dimensional) *arrays* (e.g., vectors, matrices, or tensors) of real numbers, and knowledge as functions over such data. This statement is particularly relevant as opposed to *symbolic* knowledge representation approaches, which represent data via logic formulæ (cf. section 2.2).

Despite numbers are technically symbols as well, we cannot consider arrays and their functions as symbolic knowledge representation (KR) means. Indeed, according to [192], to be considered as symbolic, KR approaches should: *(i)* involve a set of symbols, *(ii)* which can be combined (e.g., concatenated) in possibly infinite ways, following precise grammatical rules, and *(iii)* where both elementary symbols and any admissible combination of them can be assigned with meaning—i.e., each symbol can be mapped into some entity from the domain at hand. In this subsection we discuss how sub-symbolic approaches are characterised by the frequent violation of items *(ii)* and *(iii)*.

Vectors, matrices, tensors Multi-dimensional arrays are the basic brick of sub-symbolic data representation. More formally, a D-order array consists of an ordered container of real numbers, where D denotes the amount of indices required to locate each single item into the array. The i^{th} index of the array is assumed to range through the interval $1, \ldots, d_i$, so that the whole dimension of the array – i.e., the total amount of numbers therein contained – is $d_1 \times \ldots \times d_D$. In what follows, we may abuse the notation by referring to 1-order arrays as *vectors*, 2-order array as *matrices*, and higher-order arrays as *tensors*. Along this line, we may also denote by $\mathbb{R}^n$ the set of n-dimensional vectors, by $\mathbb{R}^{n \times m}$ the set of $(n \times m)$-dimensional matrices, and by $\mathbb{R}^{d_1 \times \ldots \times d_D}$ the set of $(d_1 \times \ldots \times d_D)$-dimensional tensors.

In any given sub-symbolic data-representation task leveraging upon arrays, information may be carried by both *(i)* the actual numbers contained into the array, and *(ii)* their location into the array itself. In practice, the actual dimensions $(d_1 \times \ldots \times d_D)$ of the array play a central role as well. Indeed, sub-symbolic data processing is commonly tailored on arrays of *fixed* sizes—meaning that the actual values of $d_1, \ldots, d_D$ are chosen at design time and never changed after that. For this reason, we define sub-symbolic data representation as the task of expressing data in the form of *rigid* arrays of *numbers*. Notably, such a task is *extensional* by construction, as information can only be explicitly represented.

Local vs. Distributed An important distinction, when data is represented in the form of numeric arrays, is about whether the representation is *local* or *distributed* [192]. In local representations, each single number into the array is characterised by a well-delimited meaning—i.e., it is measuring or describing a clearly identifiable concept from a given domain. Conversely, in distributed representations, each single item of the array is nearly meaningless, unless it is considered along with its neighbourhood—i.e., any other item which is "close" in the indexing space of the array, according to some given notion of closeness. So, while in local representations the location of each number in the array is quite negligible, in distributed representations it is of paramount importance.

Consider for instance the well known Iris data set[1]: it is a tabular data set where each datum can be considered as a 5-dimensional vector. There, each component of the vector is informative *per se*: it may describe e.g., the petal/sepal length/width. Conversely, consider a data set of black/white images whose resolution is $w \times h$. There, each image can be represented as a $w \times h$ matrix of numbers in the range $[0, 1]$, where each location represents a pixel and the corresponding brightness. The single pixel carries very small information when considered alone, whereas groups of contiguous pixel may describe details which are relevant for image processing.

Feature Engineering Of course, not all data is both rigid and numeric in nature. So, to fit this paradigm, data scientists designed a plethora of conversion methods to transform data from various forms (e.g., possibly non-rigid or non-numeric, when not both) into rigid arrays of numbers. In particular, when raw data is very flexible (i.e., variable in size) and very distributed, a common method consists of computing the so-called *embeddings*, i.e., fixed-size arrays synthesising the information contained into the raw data. All such methods lay under the *feature engineering* umbrella.

The ideal situation, under a data representation perspective, is when data is in *tabular* form, i.e., N instances and M features, and all features only involve numeric values. There, each instance is naturally described by a M-dimensional vector, while the whole data set is described by an $N \times M$ matrix. However, in practice, only rarely raw data fits the rigid and numeric paradigm since the very beginning. More commonly, raw data may diverge from the paradigm in several ways—possibly simultaneously. When this is the case, a number of transformations can be applied to the data to make it converge to the paradigm.

Non-numeric features A data set may involve non-numeric features, even when of tabular form. When this is the case, each single non-numeric feature may be transformed into numeric by applying a transformation to each value. The most adequate transformation heavily depends on the domain of the feature itself:

boolean features may be trivially converted into numbers via the $\{\texttt{false} \mapsto 0, \texttt{true} \mapsto 1\}$ encoding;

ordinal features may be trivially converted into natural numbers reflecting the same ordering;

categorical features may be converted into boolean features via the one-hot encoding[2], where each K-valued feature f is replaced by K binary features $f_1, \ldots, f_K$, such that $f_k = 1 \leftrightarrow f = v_k$, assuming that v_k is the k^{th} value of f;

[1]`https://archive.ics.uci.edu/ml/datasets/iris`

structured features having a **predefined** structure (e.g., dates or timestamps) can be decomposed into their components, and the same operation could be recursively repeated until no further structured feature remains;

structured features having a **variable** structure (e.g., trees) can be transformed into tensor product representations [168], provided that some upper bound to their depth exists.

Other situations may fit the cases below.

Variable-size data A data set may involve data of variable size. Consider for instance time series (e.g., samples of some phenomenon over time), or free text, or graph-like information (e.g., friendships on social networks, citations in papers). There, despite each single instance of the data set can be trivially translated into an array of numbers of some size, any two different instances from the same data set may have different sizes and internal structure.

For instance, time series can be easily modelled as T-dimensional vectors – where T is the total amount of available samples –, and the t^{th} component of the vector represents the sample at time t; free text can be represented as W-dimensional vectors – where W is the total amount of words/bigrams/trigrams/... in the text –, and the w^{th} component of the vector represents the frequency of the w^{th} word in the text (according to some ordering of words in the text); graphs can described by $N \times N$ adjacency matrices—where N is the total amount of nodes into the graph. These data representation approaches are inherently distributed and non-rigid. In fact, for any couple of different time series (possibly sampling similar phenomena), the amount of available samples may be different. Similarly, two different texts may involve different sets of words, resulting in vectors of different sizes. Finally, two different graphs (possibly describing similar situations), may involve a different amount of nodes.

Depending on the nature of the data itself, and on the particular data-analytic goal data representation is serving, data sets of such sorts can be translated into rigid form by following one of the strategies below:

draw a number of statistics on each datum (e.g., mean, standard deviation, min, max, etc.), attempting to aggregate the information therein contained: if the same amount and sorts of statistics are drawn for each datum, the data set will then become tabular;

apply a domain-changing transformation such as the Fourier transform [51] or the wavelet transform [213];

sub-sample each variable-size datum using a fixed-size sampling step; e.g., a sliding window for time series [65], or neighbourhoods of fixed sizes for graphs;

[2]An n-dimensional vector $\mathbf{x}$ of categorical values $x_1, \ldots, x_n$ where each $x_i \in \mathcal{C} = \{c_1, \ldots, c_m\}$ can always be *one-hot encoded* into a $m \times n$ matrix where the item in position i, j is 1 iff $x_i = c_j$, or 0 otherwise.

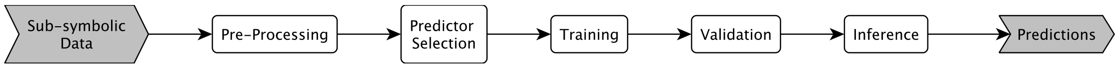

Figure 4: Major phases of the generic ML workflow.

exploit ad-hoc embeddings targetting particular sorts of data, such as GNN [206], Word2Vec [43], etc.

2.1.4 General Supervised Learning Workflow

Briefly speaking, a ML workflow is the process of producing a suitable predictor for the available data and the learning task at hand, following the purpose of later exploiting that predictor to draw analyses or to drive decisions. Hence, any ML workflow is commonly described as composed by two major phases, namely training – where predictors are fit on data – and inference—where predictors are exploited. However, in practice, further phases are included, such as data provisioning and preprocessing, as well as model selection and assessment.

In other words, before using a sub-symbolic predictor in a real-world scenario, data scientists must ensure it has been sufficiently trained and its predictive performance is sufficiently high. In turn, training requires *(i)* an adequate amount of data to be available, *(ii)* a family of predictors to be chosen (e.g., neural networks, K-nearest-neighbours, linear models, etc.), *(iii)* any structural hyper-parameter to be defined (e.g., amount, type, size of layers, K, max order of the polynomials, etc.), *(iv)* any other learning-parameter to be fixed (e.g., learning rate, momentum, batch size, epoch limit, etc.). Data must therefore be provisioned before training, and, possibly, pre-processed to ease training itself—e.g., by normalising data or by encoding non-numeric features into numeric form. The structure of the network must be defined in terms of (roughly) input, hidden, and output layers, as well as their activation functions. Finally, hyper-parameters must be carefully tuned according to the data scientist's experience, and the time constraints and computational resources at hand.

Thus, from a coarse-grained perspective, a machine learning workflow can be conceived as composed by six major phases, overviewed in Figure 4 and described into the following paragraphs.

Data set provisioning, storage, and loading Any data-driven workflow – there including supervised ML – assumes some data to be available in the first place. Technically, this implies the data has been gathered, and possibly translated in propositional/tensorial form—other than stored onto some quickly accessible repository—namely, the data set. Roughly speaking, a data set is a collection of homogenous data, often coming in the form of a single file, a folder or a database. Hence, the first actual step of any ML workflow consists of loading that data set in memory for later processing. To support such step, ML frameworks come with *ad hoc* functionalities aimed at loading the data set by reading a file from the local file system, by fetching it from the Web, or by querying a DBMS.

Data pre-processing Raw data sets are often inadequate to favour predictors' training. Hence, data set pre-processing is commonly practised to increase the effectiveness of any subsequent training phase. Pre-processing, in practice, involves a number of bulk operations to be applied to the whole data set, following several purposes, such as: *(i)* homogenise the variation ranges of the many features sampled by the data set, *(ii)* detect irrelevant features and remove them, *(iii)* construct relevant features by combining the existing ones, *(iv)* encoding non-numeric features into numeric form, and *(v)* horizontal (by row) or vertical (by column) partitioning of the data set. Purpose *(v)*, in particular, is of paramount importance, as it supports the *test set separation* – that is, a fundamental step to be performed at the very beginning of any correct ML workflow, to later enable validation and testing –, as well as splitting input-related columns from output-related ones—which, in turn, is fundamental to support training.

Predictor selection and definition Many sorts of predictors could be used in principle to perform supervised learning—e.g., neural networks, decision trees, support vector machines, etc. Unless some technical or administrative constraints exist, it is a common practice for data scientists to spend some time selecting the most adequate sort of predictor for the data and the learning task at hand. This is a common phase in virtually any ML workflow. Once a particular sort of predictor has been chosen, data scientists need a way to specify the shape the to-be-trained predictor should have. Of course, such specification should take into account the schema of the input data, as well as the schema of the expected outcomes to be produced by the predictor. The are however other aspects to be tuned, generally referred to as *hyper-parameters*.

Consider the case of neural networks as an example. Decision points in this case concern the choice of *(i)* which and how many hidden layers (of neurons) to adopt, *(ii)* how to interconnect them, and *(iii)* which activation functions to adopt for the neurons therein contained.

Training Predictors' training plays a pivotal role in ML workflows. This is the phase where predictors are fit on the available data or, in other words, automated learning actually occurs. From a workflow perspective, this phase is trivial. It simply consists of triggering the learning algorithm – which one, depends on the predictor family chosen in the previous phase –, and awaiting its termination. The overall effect is a modification of the predictor's internals to make it fit the training data.

Inference Inference is commonly the last phase of any ML workflow. Here, trained predictors are used to draw predictions on unknown data—i.e., different data w.r.t. the one used for training. In particular, given a raw datum having the same schema of the input data used for training – there including any prior pre-processing phase –, the trained predictor can be exploited to compute the corresponding prediction—even if (and especially because) the raw datum has never been observed before by that predictor. In most common cases, predictions attempt to solve classification or regression problems.

Validation Validation is the *penultimate* step of any ML workflow: it succeeds training and precedes exploitation. It is here discussed as last because it technically relies on the capability of drawing predictions via trained predictors—which is treated in the paragraph above.

Generally speaking, validation attempts to measure the predictive performance of a trained predictor, with the purpose of assessing if and to what extent it will generalise to new, unseen data. To do so, the predictor is tested against the test set—that is, a collection of unseen data, for each expected predictions exist. The discrepancy (or similarity) among the actual and expected predictions is then measured via ad-hoc scoring functions (a.k.a. measures), resulting in a performance assessment for the trained predictor. Many measures may be used to assess classifiers (e.g., accuracy, F1-score, etc.) and as many to assess regressors (e.g., MAE, MSE, R^2, etc.).

2.2 Symbolic AI Overview

(Symbolic) Knowledge representation has always been regarded as a key issue since the early days of AI, as no intelligence can exist without knowledge, and no computation can occur in lack of representation. w.r.t. arrays of numbers, symbolic KR is far more flexible and expressive, and, in particular, more intelligible. In the remainder of this subsection, we discuss why.

More precisely, here we discuss the language of FOL as a means for representing symbolic information. We choose first-order logic as it is quite general, and the other approaches can be described by either constraining or loosening the definition of FOL.

2.2.1 First-Order Logic (FOL)

First-order logic (FOL) [169] is a general-purpose logic which can be used to represent knowledge symbolically, in a very flexible way. More precisely, it allows both human and computational agents to express (i.e., write) the properties of – and the relations among – a set of entities constituting the *domain of the discourse*, via one or more *formulæ*—and, possibly, to reason over such formulæ by drawing inferences.

In table 1 the syntax of FOL is formally defined via a context-free grammar. Informally, the syntax for the general FOL formula is defined over the assumption that there exist:

- a number of constant symbols, including: a number of *functors*, denoted by monospaced symbols such as $\mathtt{f}_1, \mathtt{f}_2, \ldots$, and all real numbers;
- a number of *predicate symbols* (a.k.a. predications), denoted by italic symbols starting with a lower case letter, such as $p_1, p_2, \ldots$;
- a number of *variables*, denoted by italic symbols starting with a capital letter, such as $X_1, X_2, \ldots$.

Under such assumption a FOL formula is any expression composed by a list of quantified variables, followed by a number of *literals*, i.e., *predicates* which may or may not be

$\langle\textsf{Formula}\rangle$	$:=$	$\langle\textsf{Clause}\rangle \mid \langle\textsf{Quantifier}\rangle\langle\textsf{Formula}\rangle$
$\langle\textsf{Quantifier}\rangle$	$:=$	$`\forall'\langle\textsf{Variable}\rangle \mid `\exists'\langle\textsf{Variable}\rangle$
$\langle\textsf{Clause}\rangle$	$:=$	$\langle\textsf{Literal}\rangle \mid `('\langle\textsf{Formula}\rangle\langle\textsf{Connective}\rangle\langle\textsf{Formula}\rangle`)'$
$\langle\textsf{Connective}\rangle$	$:=$	$`\wedge' \mid `\vee' \mid `\rightarrow' \mid `\leftrightarrow' \mid `='$
$\langle\textsf{Literal}\rangle$	$:=$	$\langle\textsf{Predicate}\rangle \mid `\neg'\langle\textsf{Predicate}\rangle$
$\langle\textsf{Predicate}\rangle$	$:=$	$`\top' \mid `\bot' \mid \langle\textsf{Predication}\rangle \mid \langle\textsf{Predication}\rangle`('\langle\textsf{Arguments}\rangle`)'$
$\langle\textsf{Predication}\rangle$	$:=$	$p_1 \mid p_2 \mid p_3 \mid \ldots$
$\langle\textsf{Arguments}\rangle$	$:=$	$\langle\textsf{Term}\rangle \mid \langle\textsf{Term}\rangle`,'\langle\textsf{Arguments}\rangle$
$\langle\textsf{Term}\rangle$	$:=$	$\langle\textsf{Variable}\rangle \mid \langle\textsf{Structure}\rangle \mid \langle\textsf{Constant}\rangle$
$\langle\textsf{Variable}\rangle$	$:=$	$X_1 \mid X_2 \mid X_3 \mid \ldots$
$\langle\textsf{Structure}\rangle$	$:=$	$\langle\textsf{Functor}\rangle`('\langle\textsf{Arguments}\rangle`)'$
$\langle\textsf{Functor}\rangle$	$:=$	$\texttt{f}_1 \mid \texttt{f}_2 \mid \texttt{f}_3 \mid \ldots$
$\langle\textsf{Constant}\rangle$	$:=$	$\langle\textsf{Functor}\rangle \mid \langle\textsf{Number}\rangle \mid \langle\textsf{Boolean}\rangle$
$\langle\textsf{Number}\rangle$	$:=$	$\mathbb{R}$

Table 1: Context-free grammar for FOL. Sans-serif words among angular brackets denote non-terminal symbols, whereas symbols among single apices denote terminal symbols

prefixed by the negation operator ($\neg$)—in which case would be called negated. Each predicate consists of a predicate symbol, possibly applied to one or more *terms*. More precisely, each predicate may carry $N \geq 0$ terms. When this is the case, the predicate is said N-ary (meaning that its arity, or amount of arguments, is N). Terms may be of three sorts, namely *constants*, *structures*[3], or *variables*. Constants represent entities from the domain of the discourse. In particular, each constant references a different entity. Structures are combinations of one or more entities via one *functor*[4]. Similarly to predicates, structures may carry $M \geq 0$ terms. When this is the case, the structure is said M-ary as well. Being containers of terms, structures enable the creation of arbitrarily complex data structures combining several entities from the domain of the discourse, and treating them as a whole. Finally, variables are placeholders for unknown terms—i.e., for entities or groups of entities.

Predicates and terms are very flexible tools to represent knowledge. While terms can be used to represent or reference either entities or groups of entities from the domain of the discourse, predicates can be used to represent relations among those entities, or the properties of each single entity. There, the domain of the discourse $\mathbb{D}$ [21] is the set of all relevant entities which should be represented in FOL to be amenable of formal treatment, in a particular scenario. Should we use FOL to treat arithmetic, $\mathbb{D}$ would include the set of *natural* numbers—i.e., a symbol for each natural number. Should

[3]structures are also (and most commonly) known as "functions" into the computational logic literature. However, in computational logic, functions denote data structures rather than associations among a domain and a co-domain, as they are commonly intended. Thus, to avoid ambiguity, we choose to call then "structures" instead.

[4]functors are also known as "function symbols" in the computational logic literature

we treat calculus, $\mathbb{D}$ would include the set of *real* numbers. Should we treat kinship relationships, $\mathbb{D}$ would include a symbol for each person taken into account.

Concerning predicates, let us denote by p/N the N-ary predicate whose predicate symbol is p and whose arity is N. When $N \geq 2$, the predicate represents one or more items from the relation $p \subseteq \mathbb{D} \times \ldots \times \mathbb{D}$. So, for instance, the expression $p(t_1, \ldots, t_N)$, where all t_i are non-variable terms, denotes that the N-uple $(t_1, \ldots, t_N)$ is part of the N-ary relation subtended by p—or that, in other words, the N-ary relation p ties the entities $t_1, \ldots, t_N$ together. Similarly, the expression $\forall X_i\ p(t_1, \ldots, X_i, \ldots, t_N)$, where X_i is a variable, denotes a situation where, for each entity X_i in $\mathbb{D}$, the N-ary relation p ties the entities $t_1, \ldots, X_i, \ldots, t_N$ together. Dually, the expression $\exists X_i\ p(t_1, \ldots, X_i, \ldots, t_N)$ denotes an item of the N-ary relation p whose i^{th} item is unknown, or, in other words, an item where the first argument is t_1, the second argument is t_2, ..., and the last argument is t_N, while the i^{th} argument is arbitrary. Conversely, when $N = 1$, the predicate represents one or more items from the set $p \subseteq \mathbb{D}$. So, for instance, the expression $p(t)$, where t is a non-variable term, denotes a situation where t is an item of the set subtended by p—or that, in other words, the property p holds for the entity t. Similarly, the expression $\forall X\ p(X)$, denotes a situation where all items in $\mathbb{D}$ are items of the set p as well—or that, in other words, the property p holds for all entities in $\mathbb{D}$. Dually, the expression $\exists X\ p(X)$, denotes a situation where some item in $\mathbb{D}$ is in p as well. Finally, when $N = 0$, the predicate p represents a Boolean proposition which may or may not be true. Notably, the predicate $\top$ is always true, by construction, whereas the predicate $\perp$ is always false.

Concerning non-variable terms, let us denote by $\mathtt{f}/M$ the M-ary term whose functor is `f` and whose arity is M. When $M \geq 0$, the term is a constant and it represents some entity from $\mathbb{D}$. When $M \geq 1$, the term is a structure – i.e., a named and ordered group of terms – and it represents a complex or composite entity from $\mathbb{D}$. The actual interpretation of a structure really depends on the scenario at hand. So, for instance, in the arithmetic domain, it is possible to represent natural numbers by mimicking the Peano axioms[5] via a unary structure – e.g., `s`, for *successor* – and a constant – e.g., `z`, for *zero* – as follows: `z` represents 0, `s(z)` represents 1, `s(s(z))` represents 2, etc. Under this representation, each natural number (except `z`) is composed by its predecessor, and the successor functor `s/1`.

Structures as composite entities Structures may be used in logic to represent composite entities. Such composite entities may either be of fixed size or of variable size.

A fixed-size composite entity made up of M sub-entities may be represented in FOL via a M-ary structure. For instance, one may represent a person in terms of first name, last name, and birthdate. In that case $M = 3$, and an adequate functor is '`person`':

```
person(adam, smith, date(1723, june, 5))
```

The underlying assumption here is that dates are represented as ternary structures as well.

[5] https://www.britannica.com/science/Peano-axioms

A variable-size composite entity, in turn, may be made up of an unknown amount of sub-entities. Furthermore, two different composite entities of the same sort may be of different sizes. Consider for instances two different journeys on a map: one may involve 3 cities, and the other may involve 4 cities, yet both can be represented by *lists* of cities to be visited in a row.

Lists – and, more generally, data structures – can be represented in FOL via ad-hoc fixed-size structures, to be used recursively. In particular, a common convention is to represent *singly linked* lists of entities using:

- a binary structure, denoting element–successor couples, e.g., `cons`/2 or ./2,
- a constant, denoting the termination of the list, e.g., `nil` or [].

So, for instance, a journey from Rome to Milan, stepping through Florence and Bologna may be represented as follows:

$$\texttt{cons}(\texttt{rome},\ \texttt{cons}(\texttt{florence},\ \texttt{cons}(\texttt{bologna},\ \texttt{cons}(\texttt{milan},\ \texttt{nil})))) \tag{1}$$

whereas a journey from Rome to Naples would be as simple as:

$$\texttt{cons}(\texttt{rome},\ \texttt{cons}(\texttt{naples},\ \texttt{nil}))$$

In both cases, cities are represented by constants, whereas lists of cities are attained by combining cities into data structures—i.e., by *recursively* wrapping cities via the `cons`/2 functor, and by exploiting the constant `nil` to conclude the list.

It is worth to be mentioned that a more practical and common notation involves the exploitation of ./2 and [] instead of `cons`/2 and `nil`, respectively, where ./2 is usually written as an *infix* symbol. With this notation, the path from eq. (1), could be written as:

$$\texttt{rome}\ .\ \texttt{florence}\ .\ \texttt{bologna}\ .\ \texttt{milan}\ .\ []$$

or, equivalently:

$$[\texttt{rome},\ \texttt{florence},\ \texttt{bologna},\ \texttt{milan}]$$

Knowledge Bases From a knowledge representation perspective, knowledge bases (KB) (a.k.a. *theories*) are sets of related FOL formulæ concerning the same domain of the discourse. We denote theories as lists of dot-terminated formulæ.

For instance, a simple KB describing natural numbers may be defined as follows:

$$\begin{array}{c} natural(\texttt{z}). \\ \forall X\ natural(X) \rightarrow natural(\texttt{s}(X)). \end{array} \tag{2}$$

There, the KB is composed by two formulæ, and it aims to define the set of natural numbers by means of the unary predicate *natural*/1, the unary structure `s`/1, and the constant `z`. More precisely, the first formula states that the constant `z` is included into the set of natural numbers, by construction, whereas the second one states that, whenever some object X is in the set of natural numbers, then object $\texttt{s}(X)$ is in the same set as well. By recursively applying that formula, one may express any natural number in Peano notation.

Intensional vs. Extensional The recursive definition of natural numbers from eq. (2) is also interesting because it exemplifies the difference among *extensional* and *intensional* definitions.

In logic, one may define concepts – i.e., describe data – either extensionally or intensionally. Extensional definitions are *direct* representation of data. In the particular case of FOL, this implies defining a relation or set by explicitly mentioning the entities it involves. The $natural(\mathtt{z})$ formula from eq. (2) is a particular case of *extensional* definition of the symbol `z` as a natural number. In other words, it partially defines the *natural* set by specifying some of its items. Conversely, intensional definitions are *indirect* representations of data. In the particular case of FOL, this implies defining a relation or set by describing its elements via other relations or sets. The $\forall X\ natural(X) \rightarrow natural(\mathtt{s}(X))$ formula from eq. (2) is a particular case of *intensional* definition of any symbol of the form $\mathtt{s}(X)$ as a natural number, provided that X is a natural number as well.

Notice that the focus here is *not* on recursion. Intentional definitions must not necessarily be recursive. For instance, one may intensionally define the *child*/2 relation via the *parent*/2 relation as follows:

$$\forall X\ \forall Y\ parent(X, Y) \rightarrow child(Y, X).$$

Yet, recursive intensional predicates are very expressive and powerful, as they enable the description of infinite sets via a finite (and commonly small) amount of formulæ.

Herbrand and its ground Variables play a fundamental role in intensional KR, as they allow referencing unknown entities and tie them together via either predicates or structures. However, there exists situations – described later in this section – where the presence of variables may be troublesome. Accordingly, here we provide a number of definitions related to variable-free FOL formulæ and KB.

A term is considered *ground* if and only if *(i)* it is a constant, or *(ii)* it is a structure ant it is only composed by constant or ground arguments. In other words, a term is ground if it contains no variables, not even recursively. A predicate is ground if any term therein contained is ground as well. A formula is ground if it only contains ground predicates, and a KB is ground if it only contains ground formulæ.

We call *Herbrand universe* the set of all possible ground terms, denoted by $\mathcal{H}$. In other words, $\mathcal{H}$ is the set of all possible representations of all entities in the domain of the discourse. Given a set of constants and functor symbols, $\mathcal{H}$ can be recursively defined as the set containing: *(i)* all possible constants, and *(ii)* all structures attained by applying all possible M-ary functors to each possible M-uple of items in $\mathcal{H}$.

The Herbrand universe may easily become infinite. A single functor of arity greater than 0 – say, `f` – plus a single constant – say, `x` – are sufficient to create an infinite Herbrand universe, as the functor may be recursively applied to the constant, infinitely many times—i.e., $\mathcal{H} = \{\mathtt{x}, \mathtt{f}(\mathtt{x}), \mathtt{f}(\mathtt{f}(\mathtt{x})), \ldots\}$.

2.2.2 Relevant Subsets of FOL

Historically, most KR formalisms and technologies have been designed around either subsets or applications of the *first-order logic* (FOL). Consider, for instance, *deductive databases* [77], *description logics* [6], *ontologies* [47], *Horn* logic [123], *higher-order* logic [190], just to name a few.

Many kinds of logic-based knowledge representation systems have been proposed over the years, mostly relying on FOL – either by restricting or extending it –, e.g., on description logics and modal logics, which have been used to represent, for instance, terminological knowledge and time-dependent or subjective knowledge.

Propositional Logic Propositional logic is a very restricted subset of FOL, where quantifiers, terms, and non-atomic predicates are lacking. Hence, propositional formulæ simply consist of expressions involving one or many 0-ary predicates – i.e., *propositions* –, possibly interconnected by ordinary logic connectives. There, each proposition may be interpreted as a Boolean variable – which can either be true or false –, and the truth of formulæ can be computed as in the Boolean algebra. So, for instance, a notable example of propositional formula could be as follows:

$$p \land \neg q \to r$$

where p may be the proposition "it's raining", q may be the proposition "there's a roof", whereas r may be the proposition "the floor is wet".

The expressiveness of propositional logic is far lower than the one of FOL. For instance, because of the lack of quantifiers, each relevant aspect/event should be explicitly modelled as a proposition. Furthermore, because of the lack of terms, entities from a given domain cannot be explicitly referenced. Such lack of expressiveness, however, implies computing the *satisfiability* of a propositional formula is a *decidable* problem—which may be a desirable property in some application scenarios.

Despite propositional logic may appear too trivial to handle common decision tasks where non-binary data is involved, it turns out a number of apparently complex situations can indeed be reduced to a propositional setting. This is the case for instance of any expression involving numeric variables or constants, arithmetical comparison operators, logic connectives, and nothing more than that. Consider for instance the case of the following decision procedure aimed at classifying Iris flowers:

$$\mathit{Class} = \begin{cases} \texttt{virginica} & \leftarrow \mathit{PetalLength} > 3 \land 3 \leq \mathit{SepalWidth} < 5 \\ \texttt{versicolor} & \leftarrow \mathit{PetalLength} > 3 \land \neg(3 \leq \mathit{SepalWidth} < 5) \\ \texttt{setosa} & \leftarrow \mathit{PetalLength} \leq 3 \end{cases}$$

where *Class* is a categorical variable (whose admissible values are `virginica`, `versicolor`, and `setosa`), whereas *PetalLength* and *SepalWidth* are real-valued variables. Such formula may be rewritten in propositional logic as follows:

$$\begin{aligned}&(\mathit{virginica} \leftarrow \mathit{big\_petal} \wedge \mathit{average\_sepal})\\ &\qquad \wedge (\mathit{versicolor} \leftarrow \mathit{big\_petal} \wedge \neg \mathit{average\_sepal})\\ &\qquad\qquad \wedge (\mathit{setosa} \leftarrow \neg \mathit{big\_petal})\end{aligned}$$

where each proposition is defined as a comparison among one multi-valued variable and one constant:

- $\mathit{virginica} \equiv (\mathit{Class} = \texttt{virginica})$
- $\mathit{versicolor} \equiv (\mathit{Class} = \texttt{versicolor})$
- $\mathit{setosa} \equiv (\mathit{Class} = \texttt{setosa})$
- $\mathit{big\_petal} \equiv (\mathit{PetalLength} > 3)$
- $\mathit{average\_sepal} \equiv (3 \leq \mathit{SepalWidth} < 5)$

Description Logics: Ontologies and Knowledge Graphs Description logics (DL) are a family of subsets of FOL, generally involving some or no quantifiers, no structured terms, and no n-ary predicates such that $n \geq 3$. In other words, description logics represent knowledge by only leveraging on constants and variables, other than atomic, unary, and binary predicates.

Differences among specific variants of DL lay in which and how many logic connectives are supported, other than, of course, whether negation is supported or not. The wide variety of DL is due to the well known *expressiveness–tractability* trade-off [105, 24]. However, depending on the particular situation at hand, one may either prefer a more expressive ($\approx$ feature rich) DL variant at the price of a reduced tractability (or even decidability) of the algorithms aimed at manipulating knowledge represented through that DL, or *vice versa*.

Regardless of the particular DL variant of choice, within the scope of DL it is a common practice to call *(i)* constant terms, e.g., `abraham`, as "individuals" – as each constant references a single entity from a given domain –, *(ii)* unary predicates, e.g., $\mathit{person}(X)$, as either "classes" or "concepts" – as each predicate *groups* a set of individuals, i.e., all those individuals for which the predicate is true –, *(iii)* binary predicates, e.g., $\mathit{parent\_of}(X, Y)$, as either "properties" or "roles"—as each predicate *relates* two sets of individuals. Following such nomenclature, any piece of knowledge can be represented in DL by tagging each relevant entity with some constant (e.g., an URL), and by defining concepts and properties accordingly.

Notably, binary predicates are of particular interest as they support connecting couples of entities altogether. This is commonly achieved via subject-predicate-object *triplets*, i.e., ground binary predicates of the form $\langle \texttt{a}\ f\ \texttt{b} \rangle$ – or, alternatively, $f(\texttt{a}, \texttt{b})$ –, where `a` is the subject, f is the predicate, and `b` is the object. Such triplets allow users to *extensionally* describe knowledge in a readable, machine-interpretable, and tractable way.

Collections of triplets constitute the so-called *knowledge graphs*, i.e., directed graphs where vertices represent individuals, while arcs represent the binary properties connecting those individuals. These may explicitly or implicitly instantiate a particular *ontology*, i.e., a formal description of classes characterising a given domain, and of their relations (inclusion, exclusion, intersection, equivalence, etc.), as well as the properties they must (or must not) include.

Horn Logic Horn logic is a notable subset of FOL, characterised by a good trade-off among theoretical expressiveness, and practical tractability [113].

Horn logic is designed around the notion of *Horn clause* [88]. Horn clauses are FOL formulæ having no quantifiers, and consisting of:

- a disjunction of predicates, where only at most one literal is non-negated:

$$\neg b_1 \lor \ldots \lor \neg b_n \lor h$$

- or, equivalently (applying De Morgan rules), a disjunction among a predicate and a negated conjunction of predicates:

$$\neg(b_1 \land \ldots \land b_n) \lor h$$

- or, equivalently (applying the equivalence $\neg X \lor Y \equiv X \rightarrow Y$), an implication having a single predicate as post-condition and a conjunction of predicates as pre-condition:

$$b_1 \land \ldots \land b_n \rightarrow h$$

- often conveniently written as:

$$h \leftarrow b_1,\ \ldots,\ b_n \tag{3}$$

where $\leftarrow$ denotes logic implication from right to left, commas denote logic conjunction, and all b_i, as well as h, are predicates of arbitrary arity, possibly carrying FOL terms of any sort—i.e., variables, constants, or structures. By looking at eq. (3), it should be evident why h is often called *head* (of the clause), while the conjunction $(b_1, \ldots, b_n)$ is often called *body* (of the clause). Quantification of variables is omitted, as all variables possibly occurring in the head are assumed to be *universally* quantified, whereas all other variables possibly occurring in the body (and not in the head) are assumed to be *existentially* quantified.

So, essentially, Horn logic is a very restricted subset of FOL where:

- formulæ are reduced to clauses, as they can only contain predicates, conjunctions, and a single implication operator, therefore
- operators such as $\lor$, $\leftrightarrow$, or $\neg$ cannot be used,
- variables are implicitly quantified, and

- terms work as in FOL (there including the definition of "ground term" and "Herbrand universe").

Similarly to FOL, Horn logic KB consist of sets of Horn clauses.

Despite being very restrictive in theory, the lack of basic operators such as $\lor$, $=$, or $\neg$ can be circumvented in practice, via *meta-predicates*—i.e., predicates accepting other predicates as arguments. Circumvention in these cases steps through a smart trick: by letting the set of admissible functors *include* the set of possible predicate symbols, one may enable the representation of predicates via terms. So, a meta-predicate can be described as an ordinary predicate, accepting terms as arguments, and considering its arguments as predicates.

It is worth to be noted that Horn clauses can be read under both a *logic* and a *procedural* perspective [191]. Under a logic perspective, Horn clauses are bare implications, which can be used to define relations or sets, as in FOL. Under a procedural perspective, any Horn clause states that "to prove h, one should prove all $b_1, \ldots, b_n$ first".

2.3 eXplanable Artificial Intelligence (XAI)

Most intelligent systems today leverage on *sub-symbolic* predictive models that are trained from data through ML. The reason for such wide adoption is easy to understand. We live in an era where the availability of data is unprecedented, and ML algorithms make it possible to detect useful statistical information hidden into such data semi-automatically. Information, in turn, supports decision making, monitoring, planning, and forecasting virtually in any human activity where data is available.

However, ML is not the silver bullet. Despite the increased predictive power, ML comes with some well-known drawbacks which make it perform poorly in some use cases. One blatant example is algorithmic *opacity*—that is, essentially, the difficulty of the human mind in *understanding* how ML-based IS function or compute their outputs. This represents a serious issue in all those contexts where human beings are liable for their decision, or when they are expected to provide some sort of *explanation* for it—even if the decision has been supported by some IS. For instance, think about a doctor willing to motivate a serious, computer-aided diagnosis, or a bank employee in need of explaining to a customer why his/her profile is inadequate for a loan. In all contexts, ML is at the same time an enabling – as it aids the decision process by automating it – and a limiting factor—as opacity prevents human awareness of *how* the decision process works.

Opacity is why ML predictors are also referred to as *black boxes* into the literature. The "black box" expression refers to models where knowledge is not explicitly represented [108]. The lack of some explicit, symbolic representation of knowledge is what makes it hard for humans to *understand* the functioning of black boxes, and why they led to suggest or undertake a given decision. Clearly, troubles in understanding black-box content and functioning prevent people from fully trusting – therefore accepting – them. To make the picture even more complex, current regulations such as the GDPR [195] are starting to recognise the citizens' *right to explanation* [75]—which implicitly requires IS

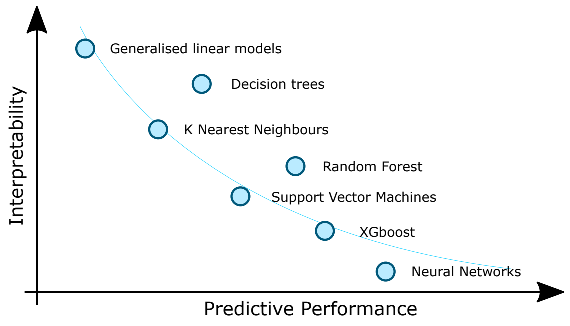

Figure 5: Interpretability/performance trade-off for some common sorts of black-box predictors.

to eventually become *understandable.* Indeed, understanding IS is essential to guarantee algorithmic fairness, to identify potential biases/problems in the training data, and to ensure that IS perform as designed and expected.

Unfortunately, the notion of understandability is neither standardised nor systematically assessed, yet. At the same time, there is no consensus on what exactly providing an *explanation* should mean when decisions are supported by a black box. However, several authors agree that not all black boxes are equally *interpretable*—meaning that some black boxes are more susceptible to understand than others for our minds. For example, Figure 5 is a common way to illustrate the differences in black-box interpretability.

Even though informal – as pointed on in [147], given the lack of ways to measure "interpretability" – Figure 5 effectively expresses why more research is need on understandability. In fact, the image essentially states how the better performing black boxes are also the less interpretable ones. This is a problem, in practice, since only rarely predictive performances can be sacrificed in favour of a higher degree of interpretability.

Nevertheless, consensus has indeed been reached about *interpretability* and *explainability* being desirable properties for intelligent systems. Hence, withing the scope of this paper, we may briefly and informally describe XAI as the corpus of literature and methods aimed at making sub-symbolic AI more interpretable for humans, possibly by automating the production of explanations.

Along this line, based on the preliminary work by Ciatto *et al.* [45, 46], and by drawing inspiration from computational logic, we let the term *interpretation* indicate

> *the subjective relation that associates each representation with a specific meaning in the domain of the problem.*

w.r.t. intelligent systems, interpretability refers to the cognitive effort required by human observers to assign a meaning to the way the system works, or motivate the outcomes it produces. Indeed, in those contexts, the notion of interpretability is often coupled with properties as algorithmic transparency (characterising approaches which are *not* opaque), decomposability, or simulatability—*predictability*, in a nutshell. In other words, interpretable systems may be understandable in the sense that humans can predict their behaviour.

As far as the term *explanation* is concerned, we trace back its meaning to the Aristotelian thought, other than the Oxford dictionary definition. In particular, the Oxford dictionary defines the term *explanation* as

> *a set of statements or accounts that makes something clear, or, alternatively, the reasons or justifications given for an action or belief.*

Thus, an explanation is an activity aimed at making the relevant details of an object clear or easy to understand to some observer.

Accordingly, the concepts of explainability and interpretability are basically *orthogonal*. However, since they are not unrelated notions, it may happen that an explanation could be given by exploiting the interpretability feature.

This is the case, for instance, of *explanation by model simplification* [179], where a poorly-interpretable model is translated into another – a more interpretable one –, having "high fidelity" [78] w.r.t. the first one. The translation process of the first model into the second one can be considered as an explanation. For instance, as we further discuss in the remainder of this paper, there exists a number of methods for *extracting* logic knowledge out of sub-symbolic predictors. In this case, the extraction algorithm is technically an explanation, as it produces (more) interpretable objects – namely, the logic knowledge – out of (less) interpretable ones—i.e., the sub-symbolic predictor.

Conversely, one may regulate the interpretability of a(n opaque) model by altering it to become "consistent" w.r.t. to some more interpretable one. In this case, no explanation is involved, yet the resulting model has a higher degree of interpretability—which is commonly the goal. For instance, as we further discuss in the remainder of this paper, there exists a number of methods for *injecting* logic knowledge into sub-symbolic predictors. In this case, the injection algorithm is the means by which opacity issues are worked around.

2.3.1 Kinds of explanation

According to the main impact surveys in the XAI area [78, 15, 28], two major approaches exist to bring explanability or interpretability features to IS, namely either *by-design* or *post-hoc*.

XAI by design. This approach to XAI aims at making intelligent systems interpretable or explainable *ex-ante*, since they are designed keeping these features as first-class goals. Method adhering to this approach can be further classified according to two sub-categories:

transparent box design containing methods supporting the creation of predictive models that are inherently interpretable, requiring no furhter manipulation;

symbols as constraint containing methods supporting the creation of predictive models – possibly including or involving some black-box components – whose behaviour is constrained by a number of symbolic and intelligible rules, usually expressed in terms of (some subset of) first-order logic.

In particular, in the remainder of this paper, we focus on methods from the second category, as it is deeply entangled with symbolic knowledge *injection*.

Post-hoc explanability. This approach to XAI aims at making intelligent systems interpretable or explainable *ex-post*, i.e., by somehow manipulating poorly interpretable pre-existing systems. Method adhering to this approach can be further classified according to the following sub-categories:

text explanation where explainability is achieved by generating textual explanations that help to explain the model results; methods that generate symbols representing the model behaviour are also included in this category, as symbols represent the logic of the algorithm through appropriate semantic mapping;

visual explanation techniques that allow the visualisation of the model behaviour; several techniques existing in the literature comes along with methods for dimensionality reduction, to make visualisation human-interpretable;

local explanation where explainability is achieved by first segmenting the solution space into less complex solution subspaces relevant for the whole model, then producing their explanation;

explanation by example allows for the extraction of representative examples that capture the internal relationships and correlations found by the model;

model simplification techniques where a completely-new simplified system is built, trying to optimise similarity with the previous one while reducing complexity;

feature relevance methods focus on how a model works internally by assigning a relevance score to each of its features, thus revealing their importance for the model in the output.

In particular, in the remainder of this paper, we focus on methods from the 'model simplification' category, as it is deeply entangled with symbolic knowledge *extraction*.

3 Symbolic Knowledge Extraction and Injection

In this section we provide broad definitions for both symbolic knowledge extraction and injection, following the purpose of drawing a line among what methods, algorithms, and technologies from the literature should be considered related to either SKE or SKI, and what should not. We do so under a XAI perspective, hence highlighting how both SKE and SKI help mitigating the opacity issues arising in data-driven AI. Then, we discuss the potential arising from the *joint* exploitation of both SKE and SKI.

Notably, we tune our definitions to comprehend and generalise the many methods and algorithms surveyed later in this paper. Indeed, in a seek for a wider degree of generality, we provide definitions committing to no particular form of symbolic knowledge, nor sub-symbolic predictor—despite many surveyed techniques come with commitments of such sorts. Hence, in what follows, we write "symbolic knowledge" meaning "any chunk of intelligible information expressed in *any* possibly sort of logic", as well as any sort of information which can be rewritten in logic form (e.g., decision trees). Similarly, we write "sub-symbolic predictor" meaning "any sort of *supervised* ML model which can be fit over numeric data to eagerly solve classification or regression tasks".

3.1 Extraction

Generally speaking, SKE serves the purpose of generating intelligible representations for the sub-symbolic knowledge a ML predictor has grasped from data during learning. Here, we provide a general definition of SKE, and we discuss its purpose and the major benefits it brings w.r.t. the XAI panorama.

Definition We define SKE as

> *any* algorithmic *procedure accepting* trained *sub-symbolic predictors as input and producing* symbolic *knowledge as output, in such a way that the extracted knowledge reflects the behaviour of the predictor with high* fidelity*.*

Notably, this definition emphasises a number key aspects of SKE which are worth to be described in further detail.

First, SKE is modelled as a class of *algorithms* – hence finite-step recipes – characterised by what they accept as input and what they produce as output.

Concerning the inputs of SKE procedures, the only explicit requirement is on *trained* ML predictors. There is no constraint w.r.t. the nature of the predictor itself, hence SKE procedures may be designed for any possible predictor family, in principle. Yet, this requirement subtends a situation where the predictor's training has already occurred, and it has reached some satisfying performance w.r.t. the task it has been trained for. Hence, in a ML workflow, SKE should occur *after* training and validation were concluded.

Concerning the outputs of SKE procedures, the only explicit requirement is about the production of *symbolic* knowledge. "Symbolic" is here intended, in a broader sense, as a synonym of "intelligible" (for the human being), hence admissible outcomes are logic formulæ as well as decision trees, or bare human-readable text.

In any case, for an algorithm to be considered a valid SKE procedure, the output knowledge should mirror the behaviour of the original predictor w.r.t. the domain it was trained for, as much as possible. This subtends some fidelity score aimed at measuring how well the extracted knowledge mimics the predictor it was extracted by, w.r.t. the domain and the task that predictor was trained for. This, in turn, implies the extracted knowledge should, in principle, act as a predictor as well, thus being queryable as the original predictor would. Thus, for instance, if the original predictor is an image classifier, the extracted knowledge should let an intelligent agent classify images of the same sort, expecting the same result. The agent may then be computational (i.e., a software program) or human, depending on whether the extracted knowledge is machine- or human-interpretable. Notably, the exploitation of *logic* knowledge as the target of SKE is of particular interest as it would enable both options.

Purpose and benefits Generally speaking, one may be interested in performing SKE to inspect the inner functioning of an opaque predictor, which should be considered a black box otherwise. However, one may also perform SKE to automatise and speed up the process of acquiring symbolic knowledge, instead of crafting knowledge bases manually.

Inspecting a black-box predictor through SKE, in turn, is an interesting capability within the scope of XAI. Given a black-box predictor and a knowledge-extraction procedure applicable to it, any extracted knowledge can be adopted as a basis to construct explanations for that particular predictor. Indeed, the extracted knowledge may act as an *interpretable replacement* (a.k.a. surrogate model) for the original predictor, provided that the two have a high fidelity score [45].

Accordingly, the application of SKE to XAI brings a number of relevant opportunities, e.g., by letting human users *(i)* study the internal operation of an opaque predictor to find, for instance, mispredicted input patterns; or correctly predicted input patterns leveraging upon some unethical decision process; *(ii)* highlight the differences or the common behaviours between two or more BB performing the same task; *(iii)* merge the knowledge acquired by various predictors – possibly of different kinds – on the same domain—provided that the same representation format is used for extraction procedures [44].

3.2 Injection

Generally speaking, SKI serves a dual purpose w.r.t. to SKE. In particular, SKI aims at letting a ML predictor keep some symbolic knowledge into account when drawing predictions. Here, we provide a general definition of SKI, and we discuss its purpose and the major benefits it brings w.r.t. the XAI panorama.

Definition We define SKI as

> *any* algorithmic *procedure affecting how sub-symbolic predictors draw their inferences in such a way that predictions are either* computed *as a function of, or made* consistent *with, some* given *symbolic knowledge.*

This definition emphasises a number key aspects of SKI which are worth to be described in further detail. Similarly to SKE, it is modelled as a class of *algorithms*. Yet, dually w.r.t. extraction, SKI algorithms are procedures accepting symbolic knowledge as input and producing ML predictors as output.

Concerning the inputs of SKI procedures, the only explicit requirement is that knowledge should be symbolic and user-provided—hence *human*-interpretable. However, since any input knowledge should be algorithmically manipulated by the SKI procedure, we elicit an implicit requirement here, constraining the input knowledge to be *machine*-interpretable as well. This implies that some formal language – e.g., some formal logic, or some decision tree – should be employed for knowledge representation, while free text or natural language should be avoided.

Along this line, another implicit requirement is that the input knowledge should be *functionally analogous* w.r.t. the predictors undergoing injection. In other words, if a predictor aims at classifying customers' profiles as either worthy or unworthy for credit, then the symbolic knowledge should encode decision procedures to serve the exact same purpose, and observe the exact same information.

Concerning the outcomes of SKI procedures, our definition identifies two relevant situations—which are not necessarily mutually-exclusive. On the one side, SKI procedures may enable sub-symbolic predictors to accept symbolic knowledge as input. SKI procedures of this sort essentially consist of pre-processing algorithm aimed at encoding symbolic knowledge in sub-symbolic form, hence enabling sub-symbolic predictors to accept them as input. In this sense, SKI procedures of this sort enable sub-symbolic predictors to (learn how to) *compute* predictions as functions of the symbolic knowledge the were fed with—assuming it has been conveniently converted into sub-symbolic form. On the other side, SKI procedures may alter sub-symbolic predictors so that they draw predictions which are *consistent* with the symbolic knowledge—according to some notion of *consistency*. SKI procedures of this sort essentially affect either the structure or the training process of the sub-symbolic predictors they are applied to, in such a way that the predictor must then keep the symbolic knowledge into account when drawing predictions. In this sense, SKI procedures of this sort force sub-symbolic predictors to learn not only from data but from symbolic knowledge as well.

In any case, regardless of their outcomes, SKI procedures fit the ML workflow in its early phases, as they may affect both preprocessing and training.

Notably, consistency plays a pivotal role in SKI, dually w.r.t. what fidelity does for SKE. Along this line, our definition subtends some consistency score aimed at measuring how well the predictor undergoing injection can take advantage from the injected knowledge, w.r.t. the domain and the task that predictor was trained for. So, for instance, if a knowledge base states that loans should be guaranteed to people from a given minority – as long as annual income overcomes a given threshold –, then any predictor undergoing injection of that knowledge base should output predictions respecting that statement—or at least minimise violations w.r.t. it.

Purpose and benefits Generally speaking, one may be interested in performing SKI to reach a higher degree of control on what a sub-symbolic predictor is learning. In fact, SKI may either incentivise the predictor to learn some desirable behaviour, or discourage it from learning some undesired behaviour. However, one may also exploit SKI to perform sub-symbolic or fuzzy manipulations of symbolic knowledge, which would be otherwise unfeasible or hard to formalise via crisp symbols. While the latter option is further analysed by a number of authors – such as [1, 103] –, in the remainder of this section we focus on the former use case, as it is better suited to serve the purposes of XAI.

Within the scope of XAI, symbolic knowledge injection is an interesting capability as it provides a workaround for the issues arising from the opacity of ML predictors. While SKE aims at reducing the opacity of predictor by letting users observe its behaviour, SKI aims at bypassing the need for transparency. Indeed, predictors undergoing the injection of *trusted* symbolic knowledge provide higher guarantees about their behaviour, which will be more predictable and comprehensible—hence less astonishing.

Accordingly, the application of SKI to XAI brings a number of relevant opportunities, e.g., by letting the human designers *(i)* endow sub-symbolic predictors with their common sense, and, therefore: *(ii)* finely control what predictors are learning, and, in particular, *(iii)* let predictors learn about relevant situations, despite poor data is available to describe them. Provided that adequate SKI procedures exist, all such use cases come at the price of handcrafting ad-hoc knowledge bases reifying the designers' common sense in symbols, and then injecting it in ordinary ML predictors.

3.3 Combining Extraction and Injection

Here we discuss the benefits arising from the *joint* exploitation of both SKI and SKE in the engineering of AI solutions. Along this line, we elicit two major expected benefits, namely: *(i)* the possibility of *debugging* sub-symbolic predictors, and *(ii)* the exploitation of symbolic knowledge as the *lingua franca* among heterogeneous hybrid systems. Accordingly, in the remainder of this sub-section, we delve into the details of these expected benefits.

3.3.1 Debugging sub-symbolic predictors

Debugging is a common activity for software programmers. It aims at spotting and fixing *bugs* in computer programs under production/maintenance. There, a bug is some unknown error contained into the program which leads to some unexpected or undesired observable behaviour of the computer(s) running that program. The whole procedure relies on the underlying assumption that computer programs are intelligible to the programmer debugging them, and that the program can be precisely edited to fix the bug.

One may consider XAI techniques as means to "debug" sub-symbolic predictors. In this metaphor, sub-symbolic predictors correspond to computer programs – despite they are not manually written by programmers, but learned from data –, while data scientists correspond to programmers. However, debugging sub-symbolic predictors is hard, because of their opacity – which makes their innner behaviour poorly intelligible for data scientists –, and because they cannot be precisely edited after training—but should be rather retrained from scratch. Accordingly, here we discuss the role of SKE and SKI to let data-science in overcoming these issues, hence allowing data scientists to debug sub-symbolic predictors.

Figure 6 provides an overview of how SKI and SKE fit the generic ML workflow. In particular, the figure stresses the relative position of both SKI and SKE w.r.t. the other phases of the ML workflow. Notably, SKI should occur before (or during) training, while SKE should occur after it. However, the figure also stresses the addition of a *loop* into an otherwise linear workflow (right-hand side of the figure). We call it the "train-extract-fix-inject" (TEFI) loop, and we argue it is a possible way to debug sub-symbolic predictors.

In the TEFI loop, SKE is the basic mechanism by which the inner functioning of a sub-symbolic predictor (i.e., 'the program', in the metaphor) is made intelligible to data scientists. The extracted knowledge may then be understood by data scientists

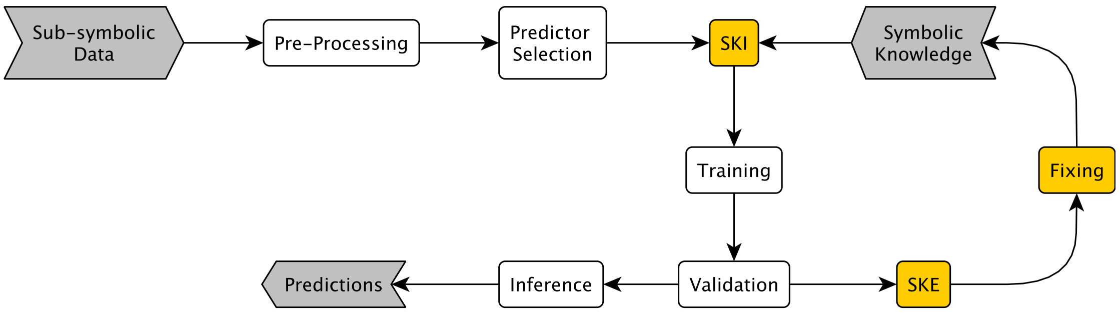

Figure 6: ML workflow enriched with SKI and SKE phases. On the right, the train-extract-fix-inject loop is represented.

and "debugged"—hence looking for pieces of knowledge which are wrong w.r.t. the data scientists expect. Then, symbolic knowledge may be precisely edited and fixed. Along this line, SKI is the basic mechanism by which a trained predictor is precisely edited to adhere to the fixed symbolic knowledge.

3.3.2 Symbolic knowledge as the lingua franca for intelligent systems

As depicted in Figure 7, intelligent systems can be suitably modelled and described as composed by several intelligent, heterogeneous, and hybrid computational agents interoperating – and possibly communicating – among each others. There, a computational agent is any software or robotic entity capable of computing, other than perceiving and affecting some given environment—be it the Web, the physical world, or anything in between. To make the overall systems intelligent, these agents should be capable of a number of intelligent behaviours, ranging from image, speech, or text recognition to autonomous decision making, planning, or deliberation. Behind the scenes, these agents may (also) leverage upon sub-symbolic predictors – possibly trained upon locally-available data –, as well as symbolic reasoners, solvers, or planners to support such sorts of intelligent behaviours. In this sense, such agents are *hybrid*, meaning that they involve both symbolic and sub-symbolic AI facilities. However, interoperability may easily become a mirage because of *(i)* the wide variety of algorithms, libraries, and platforms supporting sub-symbolic ML, other than *(ii)* the possibly different data items each agent may locally collect and later train predictors upon. Indeed, each agent may learn (slightly) different behaviours, due to the differences in the training data and in the actual ML workflow it locally adopts. When this is the case, exchange of behavioural knowledge may become cumbersome or infeasible.

In such scenarios, SKI and SKE may be enablers of a higher degree of interoperability, by supporting the exploitation of symbolic knowledge as the *lingua franca* for heterogeneous agents. Indeed, hybrid agents may exploit SKE to extract symbolic knowledge out of their local sub-symbolic predictors, and exchange (and possibly improve) that symbolic knowledge with other agents. Then, any possible improvement of the symbolic knowledge attained via interaction, may be back-propagated into local sub-symbolic

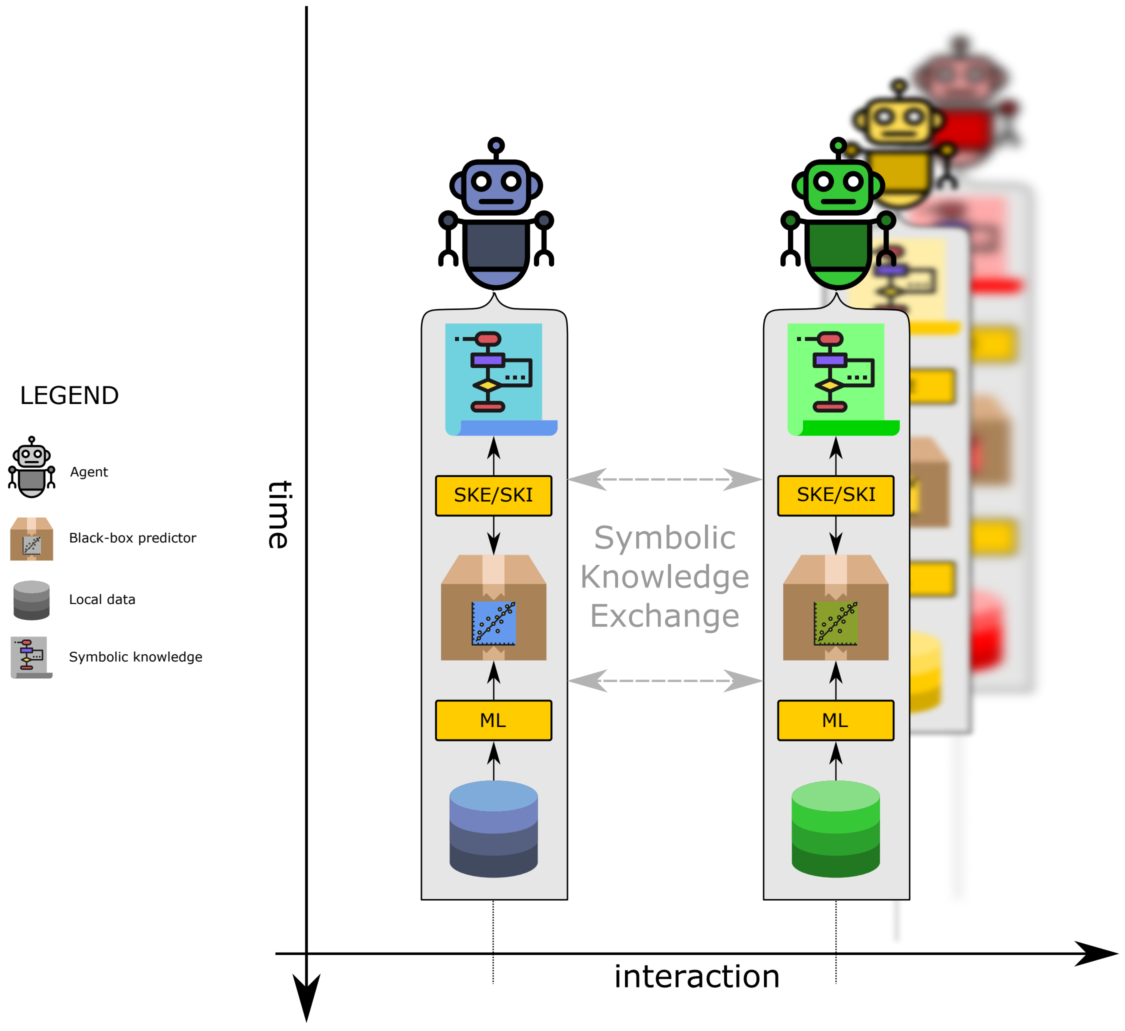

Figure 7: Conceptual representation of intelligent systems composed by many hybrid computational agents exchanging symbolic knowledge among each others. There, symbolic knowledge acts as the *lingua franca* enabling interoperability in spite of heterogeneity

predictors via SKI, hence enabling agents' behaviour to improve as well.

4 Taxonomies

In section we summarise the most relevant ways of categorising both SKE and SKI approaches. The proposed categorisations may be of interest for the readers in need of comparing or selecting SKE/SKI methods.

It is worth mentioning that our proposed taxonomies are research-driven, meaning that they have been attained be reverse-engineering, generalising, and categorising the methods surveyed in section 5. Of course, our taxonomies take into account and extend other categorization attempts from the literature, taking advantage from prior works categorising some portion of the methods discussed in section 5.

4.1 Categorisation of SKE approaches

According to [28], there are three different dimensions related to the categorisation of knowledge-extraction techniques, i.e., *(i)* the supported learning task; *(ii)* the algorithm translucency; *(iii)* the produced knowledge's shape.

Item *(i)* refers to the task performed by the extractor's underlying BB—namely, classification, regression or both of them. The majority of procedures present in the literature can be applied uniquely to BB classifiers – e.g., VI-Analysis [176, 177], C4.5 [142], TREPAN [53] and NeuroRule [164] –, whereas a small percentage of them is explicitly designed for BB regressors—e.g., RF5 [152], REFANN [163], ITER [91] and GridEx [150]. Only a few algorithms can handle both categories, e.g., SLAVE [31], ANN-DT [156], CART [25], G-REX [99]. To the best of our knowledge, at the moment all the extraction methods proposed in the literature are specifically designed for supervised learning tasks; there are no knowledge-extraction procedures tailored on unsupervised or reinforcement learning tasks.

Items *(ii)* and *(iii)* are discussed with more details in Section 4.1.1 and Section 4.1.3, respectively. In Section 4.1.2 an overview about the input types accepted by the knowledge extractors is reported. Finally, in Section 4.1.4 different kinds of output rules are explained.

4.1.1 Translucency

The translucency categorisation [3] mentioned in item *(ii)* refers to the need of the knowledge-extraction method to inspect the internal structure of the underlying BB model in order to produce the output rules. Extraction algorithms can be grouped into three categories w.r.t. their translucency:

decompositional – a.k.a. transparent or local – if the algorithm needs to know all or part of the underlying BB parameters, e.g., ANN unit biases and connection weights, or SVM support vectors;

pedagogical – a.k.a. learning-based or global – if the algorithm takes into account only the input data set and the underlying BB outputs;

eclectic – a.k.a. hybrid – if the algorithm merges features of both the aforementioned categories—e.g., by considering the BB parameters to perform a feature ranking and then apply a pedagogical extraction technique.

Decompositional approaches are less general – since they can be applied only to a very specific subset of BB models –, but also potentially more precise in terms of predictive accuracy and fidelity. On the other hand, pedagogical procedures can be adopted with any BB regardless of its kind, internal structure and complexity, hence allowing users to change the underlying model and make comparison, or to simply open a BB without awareness of its exact parameters. In this work we included the eclectic methods in the decompositional category, since they actually need to have access to the internal structure of the underlying BB. Furthermore, we included in the pedagogical category all the procedures directly applicable to learn from data, since they can be exploited to extract knowledge from BB by substituting the output features of the original data with the BB outputs.

4.1.2 Input data type

Classification and regression are supervised ML tasks described by data sets where input attributes can have different types. We categorise the knowledge-extraction algorithms reported in this work w.r.t. the following accepted input feature type:

binary if the feature can assume only two values, generally encoded with 0 and 1 (or -1 and 1, or *True* and *False*);

discrete or nominal, including integer, ordinal and binary attributes, if the feature can assume values belonging to a finite set, generally encoded as integer values;

continuous or real-valued, including integer, ordinal and binary attributes, if the feature can assume any real numeric value.

It is worthwhile to notice that extractors requiring only binary features can be applied to discrete data sets by performing a pre-processing of the discrete attributes involving a one-hot encoding phase. Analogously, extractors requiring discrete features can work with continuous attributes if these latter are first discretised. Continuous features can be converted into binary ones by performing a discretisation and then a one-hot encoding, in this exact order. In this work we report for each algorithm the accepted input as described by the authors. For instance, we explicit *continuous and discrete* if a preliminary discretisation phase is part of the algorithm; otherwise, we only indicate *discrete* if the real-valued feature discretisation should be performed before training the BB and extracting the knowledge.

4.1.3 Output knowledge shape

The knowledge shape mentioned in item *(iii)* refers to the form chosen by an extractor to present its output to human users. Decision rules [66, 93, 128] and trees [141, 142] are the most widespread human-comprehensible formats for the output knowledge, thus the majority of methods present in the literature adopt one of these. However, there are other solutions—e.g., decision tables or hierarchical rules. In every case, the conditions identifying the rules, nodes or table entries are expressed in terms of the same input and output data types used during the underlying BB training. We present in the following a brief description of each shape.

Lists The extracted knowledge can be represented as a list of rules. This list can be:

- **ordered or unordered** if the rules have to be evaluated from the first to the last – e.g., similarly to the Prolog syntax – or if the evaluation order is not relevant, respectively;
- **exclusive or overlapping** if only one rule or more of them can be applied to predict the output value of an input instances, respectively;
- **exhaustive or incomplete** if it is always possible to use the extracted rules to obtain an output prediction or if some input instances may be unpredictable, respectively.

Generally speaking, ordered lists tend to be exclusive, since the rule evaluation terminates when the first suitable rule is found. Unordered rules are more human-comprehensible, since they can be understood in isolation. The adoption of a default rule – i.e., a fixed output value to provide when there are no matching rules – makes the list always exhaustive.

The general structure of a rule list is the following:

$$\begin{gathered} if\ \texttt{condition}_1\ then\ \texttt{output}_1 \\ if\ \texttt{condition}_2\ then\ \texttt{output}_2 \\ \dots \\ if\ \texttt{condition}_n\ then\ \texttt{output}_n, \end{gathered}$$

where $\texttt{condition}_i$ is the *conditional* part (or *premise*) of the i-th rule and $\texttt{output}_i$ is the corresponding *conclusion* (or *action*). Actions are class labels in classification tasks, constant values in discretised regression tasks, or linear functions involving input variables in regression tasks. Conditions can assume various possibly equivalent formats, depending on the trade-off between human-readability and rule compactness. The simplest condition is a conjunction or disjunction of boolean literals regarding the input features. The most common formats are described in Section 4.1.4.

Decision trees Decision trees are hierarchical extensions of rule lists. Conditions are represented as internal node, whereas conclusions are the tree leaves. Generally, each

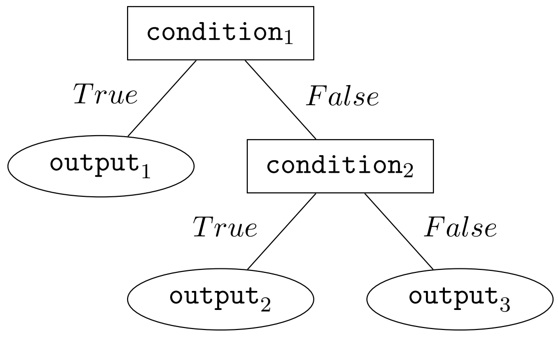

Figure 8: Example of output knowledge represented as a decision tree.

Table 2: Example of a decision table.

Variables	Rule set #1								Rule set #2			
X	0	0	0	0	1	1	1	1	–	0	1	1
Y	0	0	1	1	0	0	1	1	1	0	0	0
Z	0	1	0	1	0	1	0	1	–	–	0	1
Output	B	B	A	A	B	C	A	A	A	B	B	C

node has a condition expressed as a literal and two children—namely, one if the literal is true and the other if it is false. See Section 4.1.4 for an overview of the most common condition formats. Rule trees always can be converted into equivalent rule lists, since each path from the tree root to a leaf is a complete classification or regression rule. Rule trees are always exhaustive and are more suitable for pruning than lists. Figure 8 depicts a simple example of tree with a depth equal to 2 and 5 nodes—2 internal nodes and 3 leaves. The equivalent unordered rule list is the following:

$$\begin{aligned}&\textit{if}\ \texttt{condition}_1\ \textit{then}\ \texttt{output}_1\\ &\textit{if not}\ \texttt{condition}_1\ \textit{and}\ \texttt{condition}_2\ \textit{then}\ \texttt{output}_2\\ &\textit{if not}\ \texttt{condition}_1\ \textit{and not}\ \texttt{condition}_2\ \textit{then}\ \texttt{output}_n.\end{aligned}$$

However, it can be compacted in the following ordered list:

$$\begin{aligned}&\textit{if}\ \texttt{condition}_1\ \textit{then}\ \texttt{output}_1\\ &\textit{else if}\ \texttt{condition}_2\ \textit{then}\ \texttt{output}_2\\ &\textit{else}\ \texttt{output}_n.\end{aligned}$$

Decision tables Decision tables are concise visual rule representations specifying one or more conclusions for each set of different conditions. Tables can be exhaustive – if all the possible combinations are listed – or incomplete—otherwise. Generally, decision tables are structured as follows: there is a column (row) for each input and output

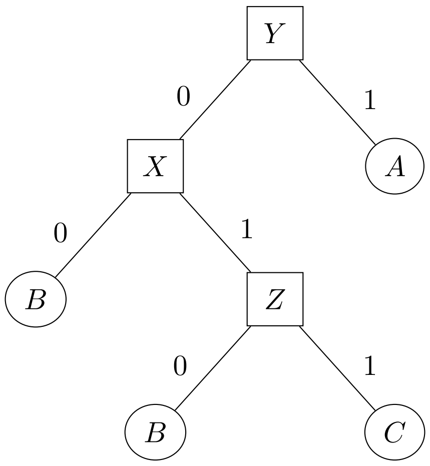

Figure 9: Decision tree equivalent to the decision table reported in Table 2.

variable and a row (column) for each rule. Each cell c_{ij} (c_{ji}) contains the value of the j-th variable for the i-th rule. Since each table cell contains a single value, this kind of representation is especially suited for classification tasks with no continuous input attributes. Decision tables can be converted into equivalent rule lists by merging all the constraints on the input values into single conditions and all those on the output values into corresponding conclusions. Exhaustive decision tables can be analogously converted into decision tables by using each constraint on input variables as split nodes.

An example of decision table with 3 binary input variables and 1 output feature – i.e., a class label – is reported in Table 2. "Rule set #1" and "Rule set #2" are semantically equivalent, but the latter is more compact since it exploits the *don't care* symbol (–). The table is exhaustive and can be converted into the rule tree depicted in Figure 9 and into the following rule list:

$$\begin{aligned}&if\ Y = 1\ then\ A\\ &else\ if\ Y = 0\ and\ X = 1\ and\ Z = 1\ then\ C\\ &else\ B.\end{aligned}$$

These conversions are not the unique possible.

Hierarchical rules Hierarchical rules are lists of rules where it is possible to have different nesting levels, similarly to rule trees. However, there is not a single starting condition – e.g., a tree root – and generally rules are nested according to specific strategies. For instance, the Re-RX extraction algorithm [161] produces as output a list of rules which conditions involve only discrete input attributes. When these rules are not enough accurate, nested sub-rules involving real-values attributes are added. An example of these rules is the following:

```
if condition₁ then output₁
if condition₂ then
    if condition₂ₐ then output₂ₐ
    if condition₂b then output₂b
if condition₃ then output₃.
```

4.1.4 Output rule conditions

Knowledge extractors produce output rules whose conditions can be expressed exploiting several formats. In the following we provide a list of the most widespread adopted formats.

Propositional rules The simplest format is a propositional condition. In this case the condition is a single literal or a conjunction/disjunction of more literals. Each literal can be considered as a predicate expressing a relationship between the value of input variable and a predefined threshold. Such relationships can be, for instance:

- **equalities** generally involving binary and discrete input variables, also in negated form—e.g., $X = 0$, $Y \neq \texttt{blue}$, $Z = \textit{true}$;
- **inequalities** involving numeric input variables, strict or non-strict—e.g., $X < \textit{upperbound}$, $Y \geq \textit{lowerbound}$;
- **set inclusion** as an extension of equalities on discrete input variables—e.g., $X \in \{1, 3, 5\}$, $Y \notin \{\texttt{red}, \texttt{green}, \texttt{blue}\}$.
- **interval inclusion** as an extension of inequalities involving numeric input variables—e.g., $X \in [\textit{lowebound}, \textit{upperbound}]$, $Y \notin [\textit{lowebound}, \textit{upperbound}]$.

M-of-N rules *M-of-N* rules are a particular sort of propositional rules where literals are grouped by n and each rule is *true* only if at least m literals out of n are *true*, with $m \leq n$. All the considerations exposed above for the relationships in the propositional case also hold for *M-of-N* rules. An example of *M-of-N* rule is: 2-*of*-{X = *false*, Y ≥ 2.5, Z ∈ [10, 20]}.

Fuzzy rules Fuzzy rules are particular propositional rules where the truth value of conditions and conclusions are not limited to 0 and 1, but can assume any value ∈ [0, 1]. This implies that conditions and conclusions can be only partially true. Fuzzy rules exploits fuzzy sets—i.e., sets whose elements have degrees of membership defined by membership functions.

Oblique rules Oblique rules have conditions expressed as inequalities between a linear combination of the input values and a threshold value. These rules create separating hyperplanes in the input feature space, thus allowing extractors to provide rules with boundaries non-perpendicular to the input space axes. Formally, an oblique rule for a problem with dimensionality d, assuming all its features are continuous, is the following:

$$if\ k_0 + \sum_{i=1}^{d} k_i x_i \lesseqgtr K\ then\ \texttt{output},$$

where k_0 is a constant term, k_i are coefficients, x_i are input variables and K is a constant threshold.

4.2 Categorisation of SKI approaches

Symbolic knowledge injection represents the set of techniques which leverage sub-symbolic computation to treat symbolic knowledge. Many different methods have been proposed to inject knowledge into sub-symbolic methods, depending on the characteristics of the information at hand or the scenario under examination. However, it is possible and helpful to categorise such approaches into three macro-categories depending on the approach used to inject the symbolic knowledge at hand: *(i) model structuring* approaches aim at mapping symbolic information into structures or set of structural constraints of sub-symbolic methods. *(ii) knowledge embedding* techniques aim at transforming the symbolic knowledge into structured data – e.g., vectors, matrices, tensors, etc. – to be injected as input(s) for the sub-symbolic model during training. *(iii) guided learning* techniques aim at translating symbolic KB at hand into constraints or components of the loss function used to train sub-symbolic models. The proposed categories represent the *integration approach* leveraged by each SKI mechanism, building the core map of SKI techniques, and are analysed more in details in Section 4.2.1 However, the proposed categorisation does not fully explain SKI techniques. Indeed, knowledge injection is characterized by many dimensions, of which the integration approach represents only one. The different purposes for which each SKI mechanism may leverage symbolic knowledge processing are analysed in Section 4.2.2, distinguishing between *knowledge manipulation* and *learning support*. Meanwhile, Section 4.2.3 covers the type of prior *input knowledge* that SKI mechanism leverage, introducing the most common solutions for each input type, their advantages and drawbacks. We then analyse the sub-symbolic models that common SKI approaches introduce, showing commonalities between most approaches (Section 4.2.4). Finally, in Section 4.2.5 we also provide an overview of the workflow to follow when dealing with the injection of symbolical knowledge into sub-symbolic models.

4.2.1 Integration Approach

Depending on the specific SKI approach analysed, the task of knowledge manipulation may be tackled differently. In this context, we call *integration approach* the strategy used by different works to manipulate the symbolic knowledge at hand. Depending on the type of knowledge available, the faced scenario, or the considered aim, there exist

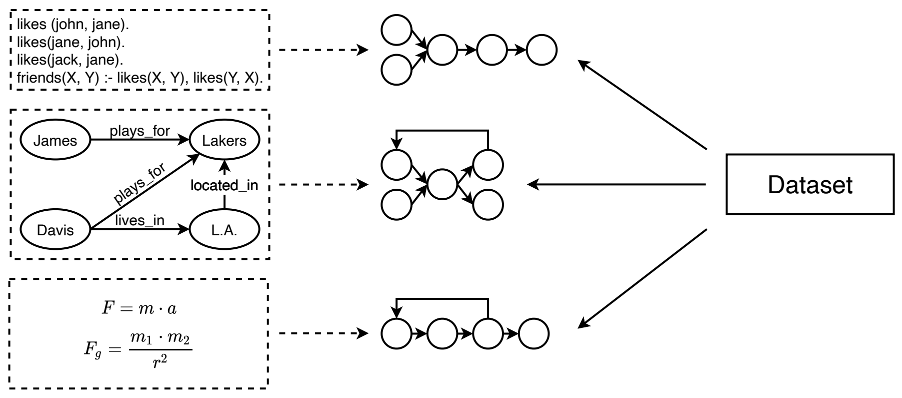

Figure 10: Workflow of model structuring SKI approaches.

multiple different ways in which such symbolic knowledge can be injected into or treated by sub-symbolic models. However, there exist some common approaches that emerge from the analysis of available SKI methods. More in details, it is possible to categorise the integration approaches of knowledge injection techniques into three macro-categories:

Model Structuring *Network structuring* methods aim at mapping the symbolic information at hand into structures of sub-symbolic models. The underlying assumption of such approaches is that the structure of sub-symbolic models partially represents the knowledge that a model can process. Therefore, modelling their anatomy enables them to satisfy or inherit the available prior knowledge. In this scenario, different knowledge basis are mapped into different sub-symbolic structures, depending on some pre-defined criteria identified by the SKI method. Therefore, the building/architectural criteria represent the core of a model structuring SKI mechanism, and its success or failure depend upon them. Figure 10 shows the general workflow characterizing model structuring approaches of SKI techniques.

Early works dealing with the integration of symbolic and sub-symbolic knowledge followed structuring approaches. Few of these works propose to map FOL clauses into weights of simple NNs, aiming at enabling sub-symbolic inference [12, 183, 138]. Building on top of these simple approaches, few works consider to introduce recurrent connections in the structure of the NN, enabling the resolution of more complex query [72, 56]. More recent approaches proposed to identify structuring criteria for ensemble of neurons or networks [55, 8, 137], overcoming limitations of previous approaches relying on the initialization of single NN weights. Structuring criteria based on distillation learning approaches have also been proposed, relying on a rule-regularized teacher NN and a student NN which is trained to imitate predictions of the teacher [89, 90]. Finally, few novel approaches propose to translate logic programs into computational graphs (NNs) using differentiable modules equivalent to logic operations, allowing for query resolution, inference and sub-symbolic manipulation [11, 114].

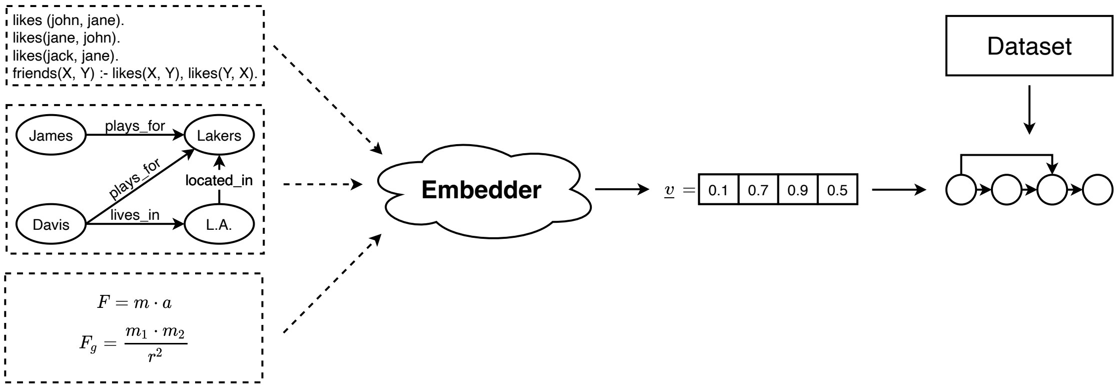

Figure 11: Workflow of SKI embedding approaches.

Knowledge Embedding *Embedding* techniques aim at transforming the unstructured symbolic information at hand into structured data—e.g., vectors, matrices, tensors, etc. Data obtained via embedding is then used as input for sub-symbolic methods to learn from it, injecting the prior knowledge. These techniques focus heavily on the manipulation of symbolic knowledge by itself, aiming at extracting relevant embeddings from it. The structured embeddings can be obtained focusing on different levels of the prior KB, from single entities-relation triplets [22] to whole knowledge basis [98]. Focusing on the manipulation of the symbolic knowledge by itself the embedding approaches are usually characterized by an underling high flexibility of the sub-symbolic model used. However, such focus usually imposes an additional burden on the way symbolic knowledge is translated into structured information, generally requiring additional learning strategies. Figure 11 shows the general workflow characterizing embedding approaches of SKI techniques.

The embedding process may be tackled following different approaches mostly depending on the type of input knowledge available. However, most popular approaches in this category rely on graph-structured prior knowledge. Many successful approaches aim at extracting and modelling relationships between entities as operations over the embedding space representing the knowledge graph at hand. Translation operations represent a very popular solution, thanks to their simplicity and effectiveness [22, 199, 172]. More complex solution may rely on the direct embedding of entities-relation triplets into tensors via some modification of rank factorization approach [132, 34].

Structured data obtained via embedding should represent as faithfully as possible the starting knowledge. Therefore, for such techniques it is common to leverage learnable embeddings that ease the representation process. Indeed, most of the analysed embedding-based SKI methods rely on some degree of guided learning (see next paragraph) in order to jointly train the sub-symbolic approach and the knowledge embeddings it leverages. Representative of such approaches are the techniques relying on graph convolution [98, 194]. Graph convolution techniques aim at extracting information directly from graph structures rather than decomposing graphs intro entities-relation triplets that are analysed individually.

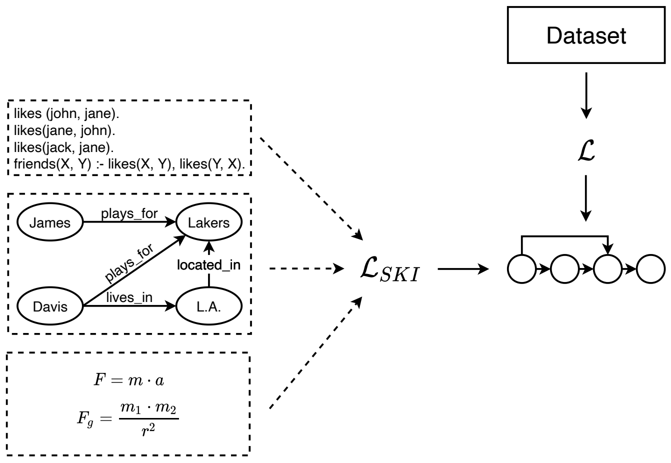

Figure 12: Workflow of SKI guided learning approaches.

Few other approaches consider to leverage expert knowledge data such as [20, 35]. These approaches may vary heavily depending on the type of expert knowledge they leverage. Interestingly, there seem to exist a lack of approaches relying on knowledge embedding that treat logic rules knowledge. This may be due to the intrinsic complexity and heterogeneity that characterise logic rules, making it cumbersome to propose embedding techniques which take in input a set of rules and produce in output a set of tensorial encodings for such rules.

Guided Learning *Guided learning* techniques aim at translating symbolic information into constraints or components of the loss function used to guide the training of sub-symbolic model. Most sub-symbolic approaches leverage custom-made loss functions to learn concepts from examples. The manipulation or constraining of such loss function allows to set some boundaries or preconceptions on the set of learnable notions. The underlying concept behind guided learning SKI approaches is that the available symbolic knowledge defines some criteria that must be satisfied by the concepts learnt from the sub-symbolic model. This logic/symbolic criteria are usually mapped into loss terms or constraints using some kind of reward/penalty functions. Such functions are designed to reward notions which adhere to the symbolic criteria, while penalising concepts which do not. Figure 12 shows the general workflow characterizing guided learning approaches of SKI techniques.

As an example, consider a classifier trained to distinguish between pictures of animal species. In this case, the prior knowledge to be injected might be represented by a phylogenetic tree, expressing relations between species. A reward/penalty function here might be used to increase the error of the classifier whenever a specie belonging to the wrong animal class is identified. Predicting a fly whenever the image of a cat is presented to the classifier brings high error during loss computation, since flies are insects while cats are mammals. On the other hand, predicting a dog when dealing with cat images brings lower error, since both are mammals. Therefore, the classifier gets more incentive to learn representations of species belonging to the same animal class closer together.

Language representation models are popular approaches belonging to this category. Indeed, leveraging guided learning it is possible to inject grammatical or syntactical knowledge into natural language processing models [76, 216, 136]. More generally, any kind of domain knowledge can be translated into loss terms, such as clinical knowledge [38, 112], algebraic expressions [2], semantic consistency [63], etc. However, FOL rules represent the most popular prior knowledge structure, probably given their natural ability of representing satisfiability criteria [57, 118, 59, 60]. Finally, it is also important to notice that guided learning may be used in conjunction with embedding or structuring techniques presented above. Indeed, some approaches rely on learnable embeddings which are trained using additional loss terms [22, 199, 187], while some others propose building custom structures of sub-symbolic models, requiring the introduction of custom loss terms [56, 144, 62].

4.2.2 Aim

We consider belonging to SKI the set of all techniques dealing with symbolic knowledge via means of a sub-symbolic approach. Indeed, such definition is helpful for identifying the set of general methods that may or may not fall into the SKI realm. However, different SKI approaches may be characterised by different aims for which symbolic knowledge is integrated into sub-symbolic models. More in details, it is possible to identify two macro-categories of goals that a SKI approach may aim to obtain:

Knowledge Manipulation Symbolic knowledge is usually treated using logic engines – e.g., ProLog, ProbLog, etc. – to extract refined knowledge or prove concepts truthfulness. However, some approaches aim at leveraging sub-symbolic models to manipulate and treat symbolic knowledge similarly to what done by symbolic engines. We define such approaches as *knowledge manipulation* SKI techniques. These approaches inject symbolic knowledge into sub-symbolic models having the aim of extracting refined symbolic information, allow complex inference, etc. Therefore, techniques belonging to this category usually produce output that can be expressed under symbolic form. The advantage of relying on sub-symbolic approaches stands in their robustness against noise. Indeed, logic engines require complete and exact information, while SKI approaches tend to generalize better, given the nature of sub-symbolic techniques. Moreover, by definition logic engines can only prove true statements, while integrated sub-symbolic approaches may evaluate also concepts which are sufficiently close to truth.

Learning Support A common approach in the SKI realm is represented by techniques which aim at injecting symbolic knowledge as helper for the sub-symbolic model at hand. Indeed, many SKI approaches consider injecting symbolic knowledge as a sort of additional information which would be otherwise complex for the sub-symbolic model to learn. The underlying idea of such approaches is that there exist some concepts that are cumbersome or troublesome to learn from examples—e.g., syntactical concepts, semantics, etc. Therefore, symbolic knowledge expressing these high-level concepts may be injected directly into the model to be trained. Techniques belonging to this category usually output a trained sub-symbolic model, characterised by learnt notions that adhere to the high-level concepts expressed by symbolic information. It is possible to notice that *learning helpers* SKI techniques focus mostly on the sub-symbolic aspect, aiming at tackling some drawbacks of sub-symbolic approaches like example hungriness, complexity, etc.

4.2.3 Input Knowledge

SKI approaches may rely on different types of input knowledge, depending on their applicative scenario and integration approach. Indeed, symbolic input knowledge may be represented using different templates. Such templates influence the operations that can be applied to the input symbolic information, enabling or hindering some approaches. It is possible to categorise the types of input knowledge, following loosely Section 2.2, and analyse their peculiarities.

Logic Rules The most common and intuitive way to represent a KB is through symbolic logic rules. Such rules represent symbolic information through logic formulae, which are usually human-readable. Logic formulae represent the most straightforward approach to define prior concepts. Therefore, such KB representation is the most popular type of input when considering guided learning SKI approaches. Indeed, guided learning techniques focus on KBs expressing criteria that a sub-symbolic model should satisfy. However, SKI mechanisms relying on logic rules KBs usually require grounded FOL logic. Groundization introduces computational burden and hinders the KB generality, and should be avoided whenever possible.

Knowledge Graphs KBs can be represented through graphs whose vertices represent entities and labelled edges represent relationships between such entities. In a sense, knowledge graphs represent the symbolic information decomposing the KB into triplets of the form entity-relation-entity. The set of all triplets, with their co-occurences, build the global KB graph. Leveraging knowledge graph as input knowledge brings representation advantages. Indeed, generally speaking, graphs are easier to treat than logic rules, especially when considering embedding-based SKI. However, knowledge graphs usually require grounded knowledge, as they identify entity instances and relations upon them. Moreover, many SKI approaches require the information available in knowledge graphs to be complete—i.e., no relation instance is missing from the graph. However, this is rarely the case, since the information used to build knowledge graphs is usually fragmented.

Expert Knowledge Some prior information might be complex to represent using either logic rules or knowledge graphs. This might be the case of physics formulae, syntactical knowledge, or any form of knowledge that is usually held by a set of human experts. Such information may be expressed in the most diverse forms – e.g., probabilistic relations, human feedback, etc. –, depending on the context and complexity. Expert knowledge variety represents its mixed blessing. Indeed, expert knowledge can be applied to most SKI methods thanks to the representation flexibility. However, expert knowledge is usually cumbersome to extract and requires some kind of human feedback, hindering its deployability.

4.2.4 Targeted Sub-symbolic Model

Analysing the works considered for this tech report it is interesting to notice that the near totality of sub-symbolic models used in SKI approaches are Neural Networks. Although it is possible for this phenomenon to be caused by sampling bias, there exist several reasons that brought authors of SKI approaches to lean preferentially towards NNs. Indeed, when analysing the integration approach of SKI methods it is possible to understand the advantages that NNs bring along.

Guided Learning NNs are sub-symbolic models that rely on Stochastic Gradient Descent optimisation techniques over custom-made loss functions to produce reasonable solutions. Therefore, *guided learning* approaches are naturally compatible with the standard NNs optimisation procedure, without requiring heavy modifications.

Model Structuring SKI approaches of this category require flexibility of the underlying sub-symbolic model to support as many architectural criteria as possible. Besides, NN represent the most broad and flexible sub-symbolic technique, thanks to their modular nature. NNs can be built from atomic components – i.e., neurons – or considering more complex block—i.e., convolution operations, etc. Such flexibility enables NNs to support a vast range of architectural criteria required by SKI approaches, making NNs the best solution when dealing with *model structuring* SKI.

Knowledge Embedding Structured data obtained from embedding of symbolic knowledge can be theoretically used as input for any sub-symbolic model. However, embedding techniques often require some guidance in learning such embeddings, which is usually provided through loss manipulation. It would be then preferable to train end-to-end the sub-symbolic model, including the embedding strategy. Such requirements are guaranteed to be met when dealing with NNs, but not with other sub-symbolic models, making NNs the preferred solution.

Therefore, given all the above considerations, it is not a hazard to consider NNs to be the natural choice of sub-symbolic model to target when dealing with SKI.

4.2.5 Workflow

Especially for readers interested in working in the SKI realm, it may be interesting to analyse the possible workflow that SKI designers should follow when implementing

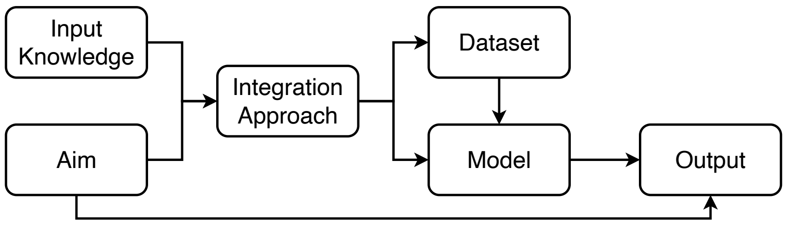

Figure 13: Workflow for implementing SKI approaches.

SKI mechanisms. Therefore, we now propose a possible SKI design workflow, paying special attention to the set of interdependencies between the components presented in Section 4.2.1 to Section 4.2.4. Figure 13 shows the general workflow to be followed when dealing with the definition and implementation of a SKI approach.

We believe the starting points of the design of SKI mechanisms to be the identification of the symbolic knowledge at hand and the aim to be obtained. Indeed, these two pillars define the type of available integration approaches to be leveraged. Considering *knowledge graphs* as input and *learning helpers* as aim strongly promotes *embedding* integration approaches. On the other hand, designers aiming to obtain *knowledge manipulation* and identifying *logic rules* as input, would probably lean strongly towards *model structuring* and *guided learning* integration approaches.

The selected integration approach limits the sub-symbolic models targetable by the SKI algorithm under construction. Moreover, the choice for the integration approach to be leveraged may influence the set of data used for the training of the model. As an example, consider *embedding* SKI approaches, which mostly rely on the construction of novel samples to be used during training. Finally, the targeted model defines the type of possible output obtained from the SKI approach. However, such output is heavily influenced also by the aim selected at the beginning of the workflow. Indeed, *knowledge manipulation* and *learning helpers* SKI techniques contemplate different semantics for the algorithm output.

5 Surveryed Methods

In this section, we survey SKE and SKI methods from the literature, classifying them accordingly to the taxonomies proposed in section 4.

Despite exhaustivity is not a goal of ours, it is worth highlighting that we gathered data following a systematic approach, aimed at increasing the reproducibility and the reach of our survey. Hence, this survey is, to the best or our knowledge, amonst the most extensive pieces of literature on this topic.

In particular, our systematic approach can be summarised as follows:

1. we select a number of sources consisting of secondary works (i.e., other similar surveys) and well-known bibliographic search engines,

2. we define a pool of queries to be performed on each search engine concerning both SKE and SKI,
3. we then analyse the bibligraphic references contained in each secondary work, looking for works to be included in our survey,
4. simultaneously, we performe each query from step 2 on each seach engine from step 1, looking for other works to be included in our survey,
5. we then skim the title and abstract of each work collected in steps 3 and 4, remonving duplicates, as well as works which are not clearly related to neither SKE nor SKI (according to the broad definitions from section 3),
6. we then read the remaning works and categorised the methods therein contained as either SKE- or SKI-related (or both),
7. finally, we cluster the surveyed works, hence eliciting the taxonomies presented in section 4.

Concerning the secondary works mentioned in step 1, we leverage on relevant surveys such as [3, 19, 28, 42, 54, 78, 80, 92, 196, 207, 220]. Conversely, concerning the bibliographic search engines mentioned in step 1, we leverage upon Google Scholar[6], Scopus[7], Springer Link[8], ACM Digital Library[9], and DBLP[10]. Finally, concerning the queries mentioned in step 2, we leverage upon the following keywords:

- ('rule extraction' OR 'knowledge extraction') AND ('neural networks' OR 'support vector machines')
- ('pedagogical' OR 'decompositional' OR 'eclectic') AND ('rule extraction' OR 'knowledge extraction')
- 'symbolic knowledge' AND ('deep learning' OR 'machine learning')
- 'embedding' AND ('knowledge graphs' OR 'logic rules' OR 'symbolic knowledge')
- 'neural' AND 'inductive logic programming'

Accordingly, in the remainder of this section, we report and categorise the surveyed methods. In particular, section 5.1 categorises methods for SKE, while section 5.2 categorises methods for SKI.

[6] `https://scholar.google.com`
[7] `https://www.scopus.com`
[8] `https://link.springer.com`
[9] `https://dl.acm.org`
[10] `https://dblp.uni-trier.de`

5.1 SKE Methods

In the following we provide a (non-exhaustive) list of SKE procedures, grouped by their translucency — i.e., decompositional techniques are discussed in Section 5.1.1, whereas pedagogical algorithms are described in Section 5.1.2. For each method we provide a categorisation according to the taxonomy presented in Section 4.1 and a brief description about the algorithm. In the case of decompositional approaches, we also specify the kind of BB predictor required by the procedure and other potential procedure-specific constraints — e.g., about the architecture or training of the underlying BB model. All these information are summarised in Table 3.

Table 3: Summary of the described knowledge-extraction algorithms.

Algorithm	Ref.	Year	Translucency	Task	Input	Rules	Shape
CART	[25]	1984	P	C+R	C+D	P	DT
ID3	[140]	1986	P	C	D	P	DT
RN	[151]	1988	P	C	D	P	L
CN2	[48]	1989	P	C	C+D	P	L
Masuoka *et al.*	[121]	1990	D (ANN)	C	C	F	L
FNES	[82]	1990	D (ANN)	C	B	F	L
MofN	[181]	1991	D (ANN)	C	D	MN	L
Berenji	[18]	1991	D (ANN)	C	C	F	L
REP	[27]	1991	P	C	C+D	P	L
ID2-of-3	[128]	1991	P	C	D	MN	DT
Horikawa *et al.*	[87]	1992	D (ANN)	C	C	F	L
Tresp *et al.*	[185]	1992	D (ANN)	R	C	P	L
SUBSET	[182]	1993	D (ANN)	C	D	P	L
VI-Analysis	[176]	1993	D (ANN)	C	C	P+MN	L
Grow	[49]	1993	P	C	C+D	P	L
C4.5	[142]	1993	P	C	C+D	P	DT
KT	[67]	1994	D (ANN)	C	D	P	L
FuNe I	[81]	1994	D (ANN)	C	C	F	L
Fuzzy-MLP	[126]	1994	D (ANN)	C	C+D	F	L
REAL	[52]	1994	P	C	B	P+MN	L
I-REP	[70]	1994	P	C	D	P	L
BRAINNE	[157]	1994	P	C	C	P	L
RULEX	[4]	1995	D (ANN)	C	C+D	P	L
Matthews and Jagielska (I)	[122]	1995	D (ANN)	C	B	F	L
Matthews and Jagielska (II)	[122]	1995	D (ANN)	C	B	F	L
RIPPER	[50]	1995	P	C	C+D	P	L
RULENEG	[139]	1995	P	C	B	P	L
NeuroRule	[164]	1996	D (ANN)	C	B	P	L

Table 3: Continued.

Algorithm	Ref.	Year	Translucency	Task	Input	Rules	Shape
DEDEC	[178]	1996	P	C	B	P	L
Yuan and Zhuang	[212]	1996	P	C	D	F	L
TREPAN	[53]	1996	P	C	B	P+MN	DT
Hong and Lee	[86]	1996	P	C	C	F	L
NeuroLinear	[165]	1997	D (ANN3)	C	C+D	O	L
RX	[159]	1997	D (ANN)	C	D	P	L
NEFCLASS	[129]	1997	D (ANN)	C	D	F	L
RF5	[152]	1997	D (ANN)	R	C	†	†
Benítez *et al.*	[17]	1997	D (ANN)	C+R	C	F	L
Ishibuchi *et al.*	[94]	1997	P	C	C	F	L
Partial-RE	[173]	1999	D (ANN)	C	C	P	L
Full-RE	[173]	1999	D (ANN)	C	C	P	L
COMBO	[102]	1999	D (ANN)	C	B	P	L
NEFPROX	[130]	1999	D (ANN)	R	D	F	L
BIO-RE	[173]	1999	P	C	B	P	L
Krishnan *et al.*	[101]	1999	P	C	C	P	DT
ANN-DT	[156]	1999	P	C+R	C+D	P	DT
MDTF	[84]	1999	P	C	C	F	L
M-of-N3	[160]	2000	D (ANN)	C	B	MN	L
Tsukimoto	[188]	2000	D (ANN)	C	C+D	P	L
Kim and Lee	[95]	2000	D (ANN4)	C	C+D	P	DT
FERNN	[162]	2000	D (ANN)	R	C+D	P+MN+O	DT
STARE	[218]	2000	P	C	C+D	P	L
MMFF	[85]	2000	P	C	C	F	L
CRED	[154]	2001	D (ANN3)	R	C+D	P	DT
Ant-Miner	[135]	2001	P	C	C+D	P	L
SLAVE	[31]	2001	P	C+R	C+D	F	L
RN2	[153]	2002	D (ANN)	R	C+D	P	L
REFANN	[163]	2002	D (ANN3)	R	C+D	P	L
Ant-Miner2	[109]	2002	P	C	C+D	P	L
DecText	[23]	2002	P	C	C+D	P	DT
REX	[117]	2003	P	C	C+D	F	L
REFNE	[219]	2003	P	C	C+D	P	L
NNRules	[167]	2004	D (ANN3)	R	C+D	P	L
RulExSVM	[68]	2004	D (SVM)	C	C+D	P	L
GEX	[116]	2004	P	C	C+D	P	L
Rabuñal *et al.*	[143]	2004	P	C	C+D	P	L
BUR	[39]	2004	P	C	C	P	L
Ant-Miner3	[110]	2004	P	C	C+D	P	L
Browne *et al.*	[26]	2004	P	C	C+D	P+MN	DT

Table 3: Continued.

Algorithm	Ref.	Year	Translucency	Task	Input	Rules	Shape
HRE	[214]	2005	D (SVM)	C	C	P	L
Barakat and Diederich	[14]	2005	D (SVM)	C+R	*	*	*
Fung *et al.*	[69]	2005	D (SVM+LC)	C	C	P	L
Chaves *et al.*	[37]	2005	D (SVM)	C	C	F	L
Torres and Rocco	[180]	2005	P	C	C+D	P+MN	DT
OSRE	[61]	2006	P	C	C+D	P	L
SVM_DT	[83]	2006	P	C	C+D	P	DT
ITER	[91]	2006	P	R	C	P	L
CoOp	[10]	2007	D (ANN)	C	B	P	L
Schetinin *et al.*	[155]	2007	D (DTE)	R	C	P	DT
Chen *et al.*	[40]	2007	D (SVM)	C	C	P	L
SQRex-SVM	[13]	2007	D (SVM)	C	C+D	P	L
HYPINV	[149]	2007	P	C	C+D	O	L
Ant-Miner+	[120]	2007	P	C	C+D	P	L
SVM+Prototypes	[133]	2008	D (SVM)	C	C	P+O	L
Re-RX	[161]	2008	P	C	C+D	P+O	H
GRG	[134]	2008	P	C	D	P	L
G-REX	[99]	2008	P	C+R	C+D	F	DT
Bader	[7]	2009	D (ANN)	C	B	P	L
ALBA	[119]	2009	D (SVM)	C	*	*	*
Lehmann *et al.*	[104]	2010	P	C	B	P	L
RxREN	[5]	2012	P	C	C+D	P	L
KDRuleEx	[158]	2012	P	C	C+D	P	TA
DeepRED	[220]	2016	D (ANN)	R	C+D	P	DT
PWL-ANN	[32]	2017	D (ANN)	R	C	P	L
MGA	[210]	2018	P	C	B	P	L
Enhanced-PWL-ANN	[33]	2020	D (ANN)	R	C	P	L
IRFRE	[198]	2020	D (DTE)	C	C	P	L
GridEx	[150]	2021	P	R	C	P	L

5.1.1 Decompositional techniques

Extracting rule lists from ANN classifiers The following techniques are explicitly designed for ANN predictors tackling classification problems. They produce as output lists of rules.

KT KT[11] [67] is a decompositional approaches suitable for NN classifiers using smooth activation functions, e.g., the sigmoid function. KT can handle binary and discrete input features – even if n-valued discrete features are converted into n binary features – and produces a list of conjunctive rules that can be *confirming* or *disconfirming*—the latter case if the rule output is negated. The algorithm heuristically searches through the rule space expanded in terms of combinations of attributes. The searching process is exponential w.r.t. the layers' number of the underlying NN, so KT performs a reduction of the search space by gathering information about ANN weights and activation functions.

SUBSET and MofN SUBSET and MofN are two decompositional techniques for NN classifiers working on discrete inputs.

SUBSET [182] produces lists of conjunctive rules and assumes that the underlying NN units are either maximally active (i.e., with activation values near 1) or inactive (i.e., with activation values near 0). Each hidden and output unit can thus be considered as a step function or a Boolean rule, so the extraction procedure consists of determining under which conditions the rule is true. The SUBSET algorithm makes a second assumption about the underlying ANN training phase, that should not significantly alter the meaning of units. SUBSET is similar to KT; its steps are the following. A simple, breadth-first subset algorithm determines if there are sets containing a single link that are sufficient to activate neurons. If so, these sets are converted into rules. Then, the search continues by increasing the subset size until the exploration of all possible subsets, removing the subsumed rules at the end of the process. A drawback of this technique is the searching cost, since the number of possible subsets is exponentially proportional to the amount of ANN connections.

MofN [181, 182] differs from SUBSET in the output rule format, since it is able to produce *M-of-N* rules. MofN is able to output fewer rules – with equivalent significance – than SUBSET, however individual rules extracted by SUBSET tend to be easier w.r.t. human interpretation. As for the time complexity of both algorithms w.r.t. the number of ANN connections, MofN results to be a cubic algorithm, whereas SUBSET is an exponential one.

VI-Analysis VI-Analysis or VIA[12] [176, 177] is a decompositional technique designed for ANN classifiers. It is suitable to be applied to any ANN – regardless of its topology, biases and weights – adopting monotonic and continuous non-linear unit transfer functions—or, at least, piece-wise monotonic and piece-wise continuous transfer functions. The algorithm accepts continuous and discrete input features and produce a list of conjunctive rules. VIA assigns a validity interval to the activation range of each NN unit – or subset of units, e.g., the only input and output units – such that the activation values of the NN lie within the found intervals. Then, a consistency check is performed

[11]KT is derived from the term "Knowledgetron", which was coined by the author to define neural networks with knowledge.

[12]VI-Analysis stands for Validity Interval Analysis

by propagating the validity intervals throughout the network and finally the intervals are converted into hypercubes boundaries for the output rules.

RULEX RULEX[13] [4] is a decompositional extraction technique designed for a particular kind of ANN classifiers—i.e., constrained error back-propagation (CEBP) multi-layer perceptrons. CEBP networks are local response ANN performing function approximation and classification similarly to radial basis function networks. CEBP hidden units are sigmoid-based locally responsive units (LRU). RULEX accepts discrete and continuous inputs and it outputs a list of conjunctive or disjunctive rules obtained directly from the LRU parameters. Each LRU creates a disjoint partition of the training set and is composed of a set of ridges—one for each input dimension. Each ridge has its own activation range and produces an appreciable output only if receives an input within its activation range. The LRU output is a thresholded sum of its ridge activations. RULEX include a pruning phase to remove irrelevant rules and rule antecedents.

NeuroRule, NeuroLinear, RX and M-of-N3 NeuroRule [164], NeuroLinear [165, 166], RX[14] [159] and M-of-N3 [160] are decompositional techniques applicable to 3-layer ANN dealing with classification tasks. All these methods are based on the discretisation of the hidden-unit (continuous) activation values. The main differences between them reside in the accepted inputs and in the produced output. NeuroRule and RX can only deal with discrete input variables and produces as output a list of conjunctive rules. The same assumption on input attributes holds also for M-of-N3, but it can extract *M-of-N* rules. On the other hand, NeuroLinear is able to work with any kind of input feature – including continuous ones – and its output is a list of oblique rules.

The four algorithms are based on the same general algorithm, summarised as follows: *(i)* train the underlying ANN by using a weight-decay backpropagation algorithm, to have network weights reflecting the connection importance; *(ii)* prune the irrelevant connections and units of the ANN, without deteriorating its predictive accuracy; *(iii)* adopt a clustering approach to discretise the activation values of the hidden units; *(iv)* generate classification rules describing the ANN outputs in terms of the discretised activation values; *(v)* generate another set of rules to describe each discretised hidden-unit activation value in terms of the ANN inputs; *(vi)* merge the two set of rules to obtain the final set of rules describing the relationship between ANN inputs and outputs.

RX is slightly different in the execution of Item *(v)*. For each hidden unit, if the number of input connections does not exceed a user-defined threshold, then rules are generated as described above. Otherwise a new ANN with an input unit for each input connection of the hidden unit is created and trained by using the discretized activation values as the target output. Then the RX algorithm is recursively applied to the new network after having introduced a new hidden layer.

[13]RULEX stands for RULe EXtraction
[14]RX stands for Rule eXtraction

Partial-RE and Full-RE Partial-RE [173] is a decompositional method applicable to ANN classifiers. It can handle continuous input features. The algorithm is based on the identification of dominant ANN connections. It sorts the incoming connections of each hidden and output unit to find individual connections able to activate the unit regardless of other connections directed to the same unit. These dominant connections are marked and never used in any further inspection. The output rule list is constructed with the marked connections. If necessary, the algorithm can use combinations of two or more unmarked connections.

Full-RE [173] is applicable under the same circumstances and outputs the same kind of rules than Partial-RE. However, it is more general since also discrete features are allowed. Full-RE extracts rules associated to certainty factors and exploits linear programming to find suitable subsets of connections to be used for the rule construction.

COMBO, CoOp and the method by Bader COMBO [102] and CoOp[15] [10] are decompositional methods to extract rules from 3-layer NN classifiers working with binary input features. The algorithms are based on the selection of input patterns that make single ANN units active or inactive. COMBO refers to these patterns as *confirming* and *disconfirming* rules, whereas CoOp adopts a different names—i.e., *coalitions* and *oppositions*, respectively. These particular patterns describe the behaviour of the underlying BB. Both algorithms search for positive and negative patterns in a constructive approach, starting from empty sets and iteratively adding one literal at a time. Literals are ordered w.r.t. their importance—i.e., the corresponding weight.

COMBO starts extracting confirming and disconfirming rules for the output layer, then it does the same for the hidden layer and finally it merges the rules to obtain descriptions of the ANN output values in term of its inputs.

CoOp works in an analogous but incremental manner, by using its coalitions and oppositions.

Bader [7] successively extended CoOp by exploiting binary decision diagrams to store the intermediate results of the extraction process.

Method by Tsukimoto Tsukimoto [188] proposed a decompositional approach applicable to ANN classifiers having monotone output function—e.g., a sigmoid function. It can handle discrete and continuous input features, even though both types require some pre-processing elaborations—e.g., one-hot encoding of the discrete attributes and normalisation of the continuous attributes. The basic idea of this procedure is to approximate the underlying ANN units with Boolean functions. Since this implies an exponential computational complexity, a pruning of the less informative terms is performed to achieve polynomial complexity. The complete explanation of this method together with all the algebraic details can be found in [188].

Methods by Masuoka et al. and Berenji Masuoka *et al.* [121] proposed a decompositional method to extract fuzzy rule lists from ANN classifiers. The algorithm requires

[15]CoOp stands for Coalitions and Oppositions

continuous input features and a peculiar underlying ANN having a 3-layer architecture and a hidden layer with either one or two units. The input layer represents the membership function of each input variable. The hidden layer represents the fuzzy operations on the input variables. The output layer represents the membership function of the output variable. A pruning phase is performed to remove connections having weights under a certain threshold.

Similarly, Berenji [18] presented a method using a specialised ANN to refine an approximately correct knowledge base of fuzzy rules. In this technique the underlying ANN is used to modify the membership functions of the rules' conditions and conclusions.

Method by Horikawa et al. and FuNe I Horikawa *et al.* [87] proposed different ANN classifiers able to automatically identify underlying fuzzy rules and to tune the corresponding membership functions. This is achieved by modifying the ANN connection weights. In this manner it is possible to decompositionally obtain a list of fuzzy rules from the ANN. This method works with continuous input attributes. The starting rule set is created by using expert knowledge or generating possible combinations of fuzzy input variables.

Halgamuge *et al.* proposed FuNe I [81], an extension of the method by Horikawa *et al.* that does not require an initial rule base. FuNe I exploits the weights of a trained ANN to generate the starting rule set.

NEFCLASS NEFCLASS[16] [129] is a decompositional method designed for ANN classifiers. The algorithm produces lists of fuzzy rules and it can be applied after having discretised – or, more precisely, partitioned in fuzzy sets – the input attributes. NEFCLASS can be resumed as follows: *(i)* initialise the rule set as empty or containing prior knowledge; *(ii)* process the training set to create all the required rules; *(iii)* process again the training set to select the best consequent for each rule; *(iv)* reduce the rule set by adopting a pruning strategy based on the classification accuracy or the coverage of the rules w.r.t. the training data.

FNES and Fuzzy-MLP FNES[17] [82] is a decompositional technique to extract lists of fuzzy rules from ANN classifiers. The algorithm can be applied with binary input features and it can be summarised as follows: *(i)* extract rules from the underlying ANN; *(ii)* assign to each extracted value a linguistic truth value (e.g., very very true, possibly true, ...) related to its degree certainty and calculated through the weighted sum of output cells; *(iii)* analogously assign a linguistic relative importance (e.g., very important, moderately important, ...) to each rule antecedent. The extraction phase considers for each possible output which inputs are discriminative in determining such output. This is achieved by inspecting the unit activation values. Initial rules are formed by using the resulting relevant input features. Input features need domain experts to be encoded in the format required by the algorithm.

[16]NEFCLASS stands for NEuro-Fuzzy CLASSification
[17]FNES stands for Fuzzy Neural Expert System

Fuzzy-MLP [126] extends FNES bu automatising this pre-processing step. Furthermore, it accepts both continuous and discrete input features.

Methods by Matthews and Jagielska Matthews and Jagielska [122] proposed two decompositional methods to extract lists of fuzzy rules from 3-layer NN classifiers. Both procedures work with binary input features. Their first technique is based on the identification of hidden units that contribute the most to a particular output pattern. This is achieved by selecting the hidden neuron with the strongest connection to the desired output unit. The important input neurons, representing fuzzy membership values for the input variables, are similarly identified. The output rule list is composed of a set of rules for each output class.

Conversely, their second method is based on estimating the overall effect of each input neuron w.r.t. all the possible outputs.

Extracting rule lists from ANN regressors The following techniques are explicitly designed for ANN predictors tackling regression problems. They produce as output lists of rules.

Method by Tresp et al. Tresp *et al.* [185] proposed a decompositional method to extract conjunctive rules from Gaussian basis function NN regressors by mapping each basis function to a conjunctive rule. The algorithm works with continuous attributes. In order to obtain reduced rule sets, a pruning phase is performed on the underlying trained ANN. The authors suggest the following strategy. Sequentially remove basis functions and conjuncts until the error increases above a threshold: *(i)* remove basis functions with smallest relative weight (considered as a measure of importance); *(ii)* then prune conjuncts by setting the largest σ_{ij} equal to infinity (effectively removing input j from basis function i; *(iii)* retrain after each removal.

RF5 and RN2 RF5[18] [152] is a decompositional approach applicable to ANN regressors trained with a particular criterion—namely, the BPQ algorithm. It can be applied to domains described by continuous input features to extract a global rule in the form of a polynomial function of the input variables. The experiments conducted by the authors showed that the RF5 algorithm can successfully discover underlying laws whose power values are not restricted to integers. In addition, it proved to have a good performance even with noisy data or irrelevant input variables.

The same authors designed another decompositional algorithm for ANN: RN2[19] [153]. RN2 can be applied to extract regression rule lists from ANN trained with multivariate data including both discrete and continuous input attributes. Output regression rules are expressed as follows: conditional parts are conjunctive logic formulæ over categorical variables, whereas action parts are polynomial equations over numeric attributes. The algorithm starts by creating a rule per data set instance, then exploit the k-means

[18]RF5 stands for Rule extraction from Facts, version 5

[19]RN2 stands for Rule extraction from Neural networks, version 2

algorithm to generalise the rules and thus reduce their number. The authors showed that it is possible to apply the C4.5 algorithm to obtain an output rule tree instead of a list.

NEFPROX NEFPROX[20] [130] is a decompositional method to extract fuzzy rules from ANN regressors. It represents an extension of the NEFCLASS procedure described in Section 5.1.1, so it is applicable under the same conditions of NEFCLASS, but for regression tasks. Also the algorithmic steps are analogous.

REFANN and NNRules REFANN[21] [163] and NNRules [167] are two decompositional extraction algorithms designed for 3-layer NN regressors having a single linear output unit. They accept continuous and discrete input variables and their output are lists of regression rules having conjunctive conditional parts. The methods are based on the activation continuous values approximation by using either a three-piece or a five-piece linear function. The input space is divided into subregions according to the different domains of the piece-wise linear functions and the following rationale: the activation values for all inputs belonging to the same subregion can be computed as a linear function of the inputs. Since the number of output rules depends on the amount of hidden neurons and the complexity of each rule depends on the number of input features, the authors also present a pruning algorithm (N2PFA) to remove redundant hidden and input units. The pruning phase has to be performed on the underlying ANN before the piece-wise approximation. NNRULES differs from REFANN in the piece-wise linear approximation, since it aims at minimising the sum of the squared errors for the training observations.

PWL-ANN and Enhanced-PWL-ANN PWL-ANN[22] [32] is a decompositional technique designed for ANN dedicated to nonlinear regression tasks. The algorithm requires continuous inputs and it is based on a piece-wise approximation of the sigmoid activation functions of the underlying ANN hidden neurons. In particular, 3-piece linear functions are adopted, since PWL-ANN takes advantage of a brute force approach to place 2 breakpoints per each hidden activation. PWL-ANN produces output rules in the form of linear equations.

Enhanced-PWL-ANN [33] is an improvement of PWL-ANN proposed by the same authors to obtain more concise rule lists with higher fidelity w.r.t. the underlying BB. The major differences reside in the breakpoint location for the piece-wise approximation. Enhanced-PWL-ANN is able to place N breakpoints to generate $(N + 1)$-piece linear functions. Then, a sampling algorithm can be chosen to locate the possible breakpoints, instead of the brute force approach. Finally, a sensitivity analysis can determine the number of breakpoints for each hidden unit. All these new features allow users to obtain

[20]NEFPROX stands for NEuro-Fuzzy function apPROXimator
[21]REFANN stands for Rule Extraction From Artificial Neural Networks
[22]PWL-ANN stands for Piece-Wise Linear Artificial Neural Network

rules with higher accuracy and fidelity and to have control on the trade-off between fidelity and comprehensibility of the extracted rules.

Extracting rule trees The following techniques produce as output rule trees and can be applied to specific BB predictors.

Method by Kim and Lee Kim and Lee [95] have designed a decompositional knowledge-extraction method applicable to 4-layer ANN accepting as input continuous features as well as discrete features. The extraction algorithm produces as output a tree of classification rules by exploiting a two-step mechanism: during the first phase (feature extraction) the first hidden layer's weights are collected as hyperplanes, whereas during the second phase (feature combination) all the extracted hyperplanes are combined using an induction tree.

FERNN FERNN[23] [162] is a decompositional approach to extract rule trees from 3-layer NN. Output rules can be oblique – if involving continuous input features – or conjunctive—if involving discrete input features. *M-of-N* rules can also be extracted. FERNN is based on the following algorithm: *(i)* identify relevant hidden units by computing their information gains; *(ii)* for each relevant unit, divide its activation values into 2 sub-intervals maximising the information gain; *(iii)* find the relevant connection from input to hidden units by checking their weights (relevant connections have large weights); *(iv)* generate rules in terms of the network inputs to distinguish the created sub-intervals of the hidden activations.

Method by Schetinin et al. Schetinin *et al.* [155] proposed a method to extract knowledge in the form of a DT from Bayesian DT ensemble classifiers accepting both continuous and discrete input features. The main idea behind the algorithm is to find the DT of the ensemble that better classifies the training data—obviously not with the same accuracy of the ensemble. The best DT is chosen during an iterative procedure aimed at finding the tree that: *(i)* covers a maximal number of training examples; *(ii)* gives the higher predictive accuracy; *(iii)* is as small as possible. The output DT can be used to extract conjunctive rules from its nodes.

CRED and DeepRED CRED[24] [154] is a decompositional technique to extract rule trees from 3-layer NN. It accepts both continuous and discrete input attributes and can be applied to classification tasks and regression tasks, even if in this latter case it introduces a discretisation in the output values. CRED decomposes a neural network by building a decision tree for each output unit of the underlying ANN. These DT describe the ANN outputs w.r.t. the hidden units. Successively, CRED performs an analogous step to build trees relating the ANN inputs with the previous DT splits. Finally, a merging phase is executed to obtain the output rules. Pruning procedures are applied

[23]FERNN stands for Fast Extraction of Rules from Neural Networks
[24]CRED stands for Continuous/discrete Rule Extractor via Decision tree induction

after each step to remove useless literals and specialised rules included in other more general rules.

DeepRED[25] [220] is an extension of CRED without constraints on the number of hidden layers of the underlying ANN. The algorithm adopts the basic idea of CRED to iteratively produce rule trees for each layer of the network, from the output layer to the input layer. After having extracted intermediate rules for each ANN layer, all the rules are merged to obtain final rules describing the ANN outputs in terms of its inputs. During the merging phase, redundant or conflicting rules and terms are removed.

Other decompositional procedures In the following are summarised other interesting decompositional approaches that are not included in the groups above.

Method by Benítez et al. Benítez *et al.* [17] proposed a decompositional procedure to extract fuzzy rule lists from ANN. It can be applied with continuous input features to tackle classification problems. The main idea behind the algorithm is to convert each ANN unit to a fuzzy rule equivalent to the corresponding neuron function. The extraction procedure inspects activation values, connection weights and neuron thresholds. Since the extracted rules are not at first in a well-readable format, the authors proposed a method to rewrite them in a more human-comprehensible fashion. The rewriting phase exploit a special operator described in [17] that describes the ANN internal behaviour.

RulExSVM RulExSVM[26] [68] is a decompositional method for SVM classifiers that exploits the underlying model's support vectors. It works with discrete and continuous input features and it produces lists of conjunctive rules. Output Rules are generated by creating hyper-rectangles whose vertices are defined by finding the intersection between support vectors and the separating hyper-plane. After their creation, hyper-rectangles are refined to exclude instances belonging to other classes, then the redundant ones are pruned.

Method by Barakat and Diederich Barakat and Diederich [14] proposed a decompositional method for extracting rules from SVM. It works with continuous input features and is based on the selection of the training instances corresponding to support vectors, that are used to compose a new data set. The class labels of this data set instances are replaced with the SVM predictions and then the data set is used to train an interpretable model—e.g., a decision tree. Output rules are then constructed by analysing the interpretable model. The extracted knowledge's shape depends on both the adopted interpretable model and inspection technique, so rule lists can be produced as well as rule trees. The authors do not make any assumption on the task to solve, so the method can be applied to both SVM classifiers and regressors.

[25]DeepRED stands for Deep neural network Rule Extraction via Decision tree induction
[26]RulExSVM stands for Rule Extraction from Support Vector Machines

Method by Chaves et al. Chaves *et al.* [37] proposed a decompositional method applicable to extract fuzzy rules from SVM classifiers accepting continuous input variables. Their algorithm can be summarised as follows: *(i)* find the support vectors' projections in the coordinate axex (each support vectors has n projections if input space has dimension n); *(ii)* construct triangular fuzzy sets for each coordinate; *(iii)* for each support vector projection in a given coordinate, calculate the membership degree of each fuzzy set and select the one with maximum membership degree; *(iv)* generate an output rule for each support vector The number of fuzzy set to be created is a user-defined parameters. Each set has equal domain size. Output rules are evaluated via their accuracy and coverage.

SQRex-SVM SQRex-SVM [13] is an technique to decompositionally extract classification conjunctive rule lists from SVM by following a separate-and-conquer strategy. It can handle binary, discrete or continuous input attributes and it builds the output rules by using the support vectors of the underlying SVM as well as the input feature importance. The importance of a feature is calculated w.r.t. its classification discriminative power by measuring the interclass separation. SQRex-SVM uses only example becoming true positive or true negative support vectors and it extracts rules only from true positive support vectors (even if true negative ones are exploited to refine the output rules). The algorithm involves both a pre-pruning and a post-pruning step to obtain a limited number of generalised rules.

Methods by Fung et al. and Chen et al. Fung *et al.* [69] proposed a decompositional method to extract conjunctive rules from linear classifiers, including SVM. It requires continuous inputs and it is classified as a decompositional method since it requires an underlying model providing a linear decision boundary. The iterative algorithm is based on the extraction of rules by solving a computationally-inexpensive constrained optimisation problem. Each rule derives from the creation of a hypercube having one vertex on the separating hyperplane and lying completely above or below that hyperplane. Since at every iteration several hypercubes may satisfy these criteria, different strategies are available to select the optimal hypercube—namely, volume maximisation (the biggest hypercube is the best one), or point coverage maximisation (the best cube is the one containing the largest amount of training samples). The extracted rules are non-overlapping but only asymptotically exhaustive.

Chen *et al.* [40] proposed another decompositional procedure based on hypercube creation and applicable under the same constraints of that presented by Fung *et al.* The main difference in the two extraction algorithms is that Chen *et al.* used the support vectors of the SVM as hypercube vertices.

SVM+Prototypes and HRE SVM+Prototypes [133] is a decompositional techniques suitable for SVM classifiers. This procedure accepts continuous input features and produces lists of conjunctive or oblique rules. Its core idea is to use the underlying SVM output decision function and a K-means clustering to determine prototype vectors for each class. The prototype vectors are then combined with support vectors to define ellipsoids in the input space that are finally converted into output *if-then* rules.

HRE[27] [214] is an approach similar to SVM+Prototypes in its algorithm and applicability. However, HRE can only outputs conjunctive rules. The main idea of HRE is to find prototype vectors for each class and then creating and expanding small hyper-rectangles around these prototypes until a stopping criterion is met. HRE stopping criteria are focused on the control of the hyper-rectangle dimension—hence, the quality of extracted rules.

ALBA ALBA[28] [119] is a decompositional technique to extract knowledge from SVM by using active learning. The algorithm focuses on input space areas most affected by noise—i.e., regions near the SVM decision boundary marking the transition between classes. ALBA exploits the underlying SVM to obtain the class labels of the training set instances and of a certain number of random generated samples. These latter samples are placed around the support vectors, since the authors demonstrated that only samples close to the decision boundary are able to improve the classification performance of the rules extracted by ALBA. Since an external rule induction algorithm is adopted to extract rules from the training set and random generated instances, the required input type and the output knowledge form depend on the external algorithm.

IRFRE IRFRE[29] [198] is a decompositional method applicable to DT ensemble classifiers. It accepts continuous and binary input features and produces conjunctive rule lists. IRFRE can be summarised as follows: *(i)* build a random forest classifiers by exploiting the CART algorithm; *(ii)* each tree node of the random forest is associated to a boolean condition and each path from root to leaves is a conjunctive rule; *(iii)* an evolutionary algorithm is adopted to identify the optimal combination of rules. The optimisation of the rules takes into account accuracy and interpretability.

5.1.2 Pedagogical techniques

Extracting rule lists from BB classifiers The following techniques are explicitly designed for BB predictors tackling classification problems. They produce as output lists of rules.

RN RN[30] [151] is a pedagogical method to produce lists of conjunctive rules as described in the following. This procedure works with discrete inputs and is focused on medical diagnostics, so each rule has the following form:

$$\text{if } D_1, D_2, \ldots, D_n \text{ then } S_1, S_2, \ldots, S_m, \tag{4}$$

[27]HRE stands for Hyper-rectangle Rule Extraction
[28]ALBA stands for Active Learning-Based Approach
[29]IRFRE stands for Improved Random Forest-based Rule Extraction
[30]RN stands for Rule extraction from Neural networks

where D_i is a disease and S_j is a symptom. Since both condition's diseases and action's symptoms can appear in either positive or negative form, output rules can express their presence as well as their absence. The rules are built via a constructive approach, starting with single symptoms able to "turn on" a disease and iteratively adding other symptoms while checking if the disease remains present. This method is adopted to explain single outcomes of the underlying BB and its rules tend to be overlapping.

RxREN RxREN [5] is a pedagogical technique directly applicable to data sets containing discrete input attributes as well as continuous ones. It performs a two-phase extraction that as a first step simplifies the underlying BB model – e.g., by pruning a neural network or a decision tree – and then constructs and refines the classifications rules for each class. The output rules' refinement consists in: *(i)* the pruning of trivially true conditions – i.e., $X > V_{min}$ or $X < V_{max}$, where X is an input variable and V_{min}, V_{max} are the minimum and maximum values for X, respectively –, *(ii)* the removal of the rule antecedents that are not discriminating in the output class prediction, and finally *(iii)* the tuning of the rules to enhance the classification of outliers and misclassified data set instances. RxREN produces as output lists of *if-then* rules.

Rule-extraction-as-learning Rule-extraction-as-learning [52] (REAL) is a pedagogical knowledge-extraction method applicable to BB classifiers accepting binary input features. It provides as output a list of generalised *M-of-N* or conjunctive rules learnt during a process driven by sampling and queries. The generalisation of the rules consists of dropping non-discriminative antecedents from each rule. The authors suggest different stopping criteria for their algorithm—e.g., reaching a user-defined predictive accuracy threshold w.r.t. a test set previously separated from the training data or after a certain number of iterations do not produce new rules.

Re-RX Re-RX[31] [161] is a pedagogical algorithm accepting both continuous and discrete inputs and producing rules which conditions on continuous attributes are disjoint by those involving discrete variables. The algorithm works as follows: *(i)* rules with discrete attributes are generated to explain the ANN classification process; *(ii)* if the accuracy of a rule is not satisfactory, Re-RX recursively refines it by generating hierarchical sub-rules including other discrete features or oblique sub-rules involving continuous variables. The recursive and hierarchical nature of this method assures that a first discrimination between the generated rules always depends on discrete features only. Continuous ones are uniquely considered in refined, nested sub-rules.

CN2 CN2 [48] is a pedagogical approach to extract list of conjunctive rules from BB performing classification tasks. Besides discrete inputs, it also accepts continuous attributes, since it performs a discretisation step. The basic idea of CN2 is to iteratively search for a conjunctive rule (involving input attributes) that covers a large number instances of a single class and few of other classes. Then, all the training samples

[31]Recursive Rule eXtraction

covered by the rules are removed from the training set and the algorithm search for a new rule, until there are satisfactory rules to extract. CN2 produces an *ordered* rule list. This means that a rule is evaluated only if the preceding are not satisfied, so the rules' order is always relevant.

REX REX[32] [117] is a pedagogical technique applicable to BB classifiers. It can handle discrete and continuous attributes and exploits an evolutionary algorithm to extract a set of fuzzy rules describing the BB behaviour. REX's individuals represent sets of fuzzy rules and fuzzy sets related to them. REX uses triangular fuzzy sets, that can be active or inactive. Each set is defined by a real value, corresponding to its central point. The fuzzy set begins (ends) at the central point of the preceding (successive) active set. Individuals are evaluated by taking into account the number of correctly classified, misclassified and unclassified training examples, as well as the complexity of both the single rules and the rule set. The algorithm stopping criteria are: executing a maximum number of steps, having no progress for a certain number of steps, reaching a user-defined threshold during the evaluation for the best individual.

GEX GEX[33] [116] is a pedagogical genetic algorithm to extract lists of conjunctive rules out of BB classifiers. It can accept both continuous and discrete input attributes. The procedure acts as a general approach of a genetic algorithm, but differs on some details. For example, GEX assumes the existence of several subpopulations evolving on islands. There is an island for each output class and each subpopulation is devoted to search rules for one specific class. GEX output rules – represented by the individuals – can be overlapping – i.e., a data set instance can be covered by more than one rule – and non-exhaustive—i.e., some data set instances can have no rules to cover them.

Method by Rabuñal et al. Rabuñal *et al.* [143] proposed another genetic algorithm similar to GEX. They share the same peculiarities—i.e., they are pedagogical methods for regressors and both work with the same kind of input attributes to produce rule lists. The main difference is that this method creates artificial examples to augment the data set until the extracted rule set reaches a user-defined accuracy threshold.

DEDEC DEDEC [178] is a pedagogical knowledge-extraction approach based on a ranking procedure of the BB inputs according to their contribution to the BB outputs. It outputs classification rule lists and works on binary input features. The authors propose the application of this method to ANN, where the feature importance can be deduced by the network weights, however the algorithm may be generalised and applied to any BB if there are some alternative ways to obtain such information. DEDEC can be summarised as follows: *(i)* find an input feature ranking; *(ii)* cluster the ranked input; *(iii)* use these clusters to generate a set of binary rules describing the relationship between the cluster attributes and the BB outputs.

[32]REX stands for Rule EXtraction
[33]GEX stands for Genetic-Rule EXtraction

BIO-RE BIO-RE[34] [173] is a pedagogical method that can be applied uniquely to simple binary classification tasks and in presence of binary input attributes to obtain lists of conjunctive rules. BIO-RE performs the following steps: *(i)* obtain the underlying model output prediction of each possible input attributes' pattern; *(ii)* generate a truth table by using the input patterns and the corresponding outputs; *(iii)* use the truth table to generate Boolean functions. Since the algorithm explores all the possible input configurations and works with binary data, it can be applied only if the problem at hand has a small number of inputs and the feature binarisation does not significantly degrade the performance of the underlying BB.

REP, Grow, I-REP and RIPPER REP, Grow, I-REP and RIPPER are pedagogical approaches for extracting knowledge in the form of conjunctive rule lists from BB classifiers. They all accept continuous as well as discrete attributes, except I-REP that can handle only the latter.

REP[35] [27] requires the training set splitting into two different subsets—namely the *growing set* and the *pruning set*. A relational learning algorithm is used to describe all the examples belonging to the growing set. The resulting knowledge is then pruned by removing and simplifying rules while monitoring the predictive accuracy against the pruning set. A decrease in such accuracy stops the pruning phase. The algorithm proved to be accurate also in presence of noise, however it presents some drawbacks, e.g., *(i)* its efficiency in terms of time complexity, *(ii)* the intrinsic disadvantages of the growing/pruning splitting (relevant training patterns should be adequately represented in both subsets), *(iii)* the intrinsic disadvantages of the *separate-and-conquer* strategy.

Grow [49] can partially address these limitations – including an enhancement in the REP time complexity – by designing a different method summarised as follows: *(i)* use the growing set to find an overfitting rule base; *(ii)* generalise each rule so that the error on the growing set decreases (or increases the least) and add the generalisations to the rule base; *(iii)* return the extracted knowledge by greedily selecting from the rule base the rules that minimise the error on the pruning set. In case of equivalent rules, the one with lowest error w.r.t. the growing set is chosen. If this is not sufficient to choose a unique rule, the shortest is picked.

I-REP[36] [70] has been presented as an extension of the REP algorithm aimed at overcoming its major limitations (even though it is not able to work with continuous inputs or multi-class classification tasks). Its core idea resides in pruning each rule right after its generation, instead of pruning the whole rule set at the end of the extraction process.

RIPPER [50] is an extension of I-REP supporting continuous attributes, missing data and multi-class classification. It exploits a variation of I-REP (named by the authors I-REP*) consisting in different pruning mechanisms and stopping criteria. Furthermore, RIPPER introduces a final rule optimisation step followed by the optional addition of

[34]BIO-RE stands for Binarized Input-Output Rule Extraction
[35]REP stands for Reduced Error Pruning
[36]I-REP stands for Incremental REP

other rules obtained by applying I-REP*. This optimisation/addition procedure may be repeatedly applied k times and in this case the authors name the algorithm RIPPERk.

BRAINNE BRAINNE[37] [157] is a pedagogical extraction procedure applicable to classification tasks. It accepts any kind of input features and produces lists of conjunctive and disjunctive rules (the latter may contain conjunctive formulæ as antecedents). The authors designed BRAINNE to be applied to ANN in the following fashion: the underlying ANN having n inputs and m outputs is converted into a new ANN with $n + m$ inputs and m outputs. Then, the new network is trained and the input feature importance is deduced by observing the corresponding weights of the original ANN w.r.t. those of the augmented one. The most important features are those which weights are subjected to slight variations. The strategy adopted by the basic version of BRAINNE relies on the training example partition into subsets of instances that can be classified with single conjunctive rules (supervised BRAINNE). The authors suggest several improvements to the basic BRAINNE algorithm: *(i)* using a 2-layer hybrid network, with unsupervised learning at the first layer and supervised learning at the second, to directly extract disjunctive rules without relying on support networks; *(ii)* monitoring the prediction error against a separate test set to perform an early stopping of the training, resulting in a better generalisation power of the extracted rules; *(iii)* applying a progressive extension of the continuous features' bounds while defining the output rules. The algorithm is classified as pedagogical since we believe that it is possible to substitute the underlying ANN with other BB having similar capabilities.

RULENEG and OSRE RULENEG [139] is a pedagogical approach to extract classification rule lists from BB working with binary input data. It adopts a constructive approach, since it creates a conjunctive rule per input pattern and a new rule is generated only when the extracted rule set is not able to correctly classify a training pattern. When a pattern is misclassified, a new rule is created as a conjunction of the inputs attributes influencing the output class prediction. These attributes are selected by performing consecutive negation of each input value and checking via the underlying BB if the predicted class changes.

OSRE [61] is an extension of the RULENEG accepting continuous and discrete input other than binary ones. These kinds of input variables are encoded with a procedure called *1-from-N* and described in [61].

REFNE and STARE REFNE[38] [219] and STARE[39] [218] are two pedagogical methods for extracting knowledge out of BB models. Authors designed these procedures to be applied to NN ensembles and single ANN, respectively, however they can be applied to any kind of BB classifier. Both methods accept discrete and continuous input features, since they apply a discretisation technique, and output a list of conjunctive rules. They

[37]BRAINNE stands for Building Representations for AI using Neural NEtworks
[38]REFNE stands for Rule Extraction From Neural network Ensemble
[39]STARE stands for STAtistic-based Rule Extraction

can be summarised as follows: *(i)* generate a data set by exhaustive or random sampling of the input space; *(ii)* discretise the created data set and adopt the underlying BB to predict corresponding labels; *(iii)* extract rules using the data set and the predicted labels. Rules are created by searching for values that assigned to single attributes, then pairs and finally triples of attributes can classify all the data set samples as belonging to the same class.

BUR BUR[40] [39] can be adopted to pedagogically extract conjunctive rules from BB classifiers. BUR requires binary or continuous input attributes. The approach consists of two phases—namley, a learning phase and a pruning phase. Pruning is kept separate from learning to have better control on the trade-off between output rule understandability and classification accuracy. During the learning phase, a rule set that maximally approximates the underlying model predictions is learned. Then, during the pruning phase, rules are dropped and simplified to enhance the human-comprehensibility of the output list without hindering its classification accuracy.

GRG GRG[41] [134] is a pedagogical technique to extract lists of conjunctive rules out of BB classifiers working with discrete input features. The algorithm is defined as "greedy" because, at each iteration, it generates the best classification rules w.r.t. the training sample coverage, the number of input attributes involved in the rule condition, and the input space coverage. Since training samples that are covered by new rules are removed before generating other rules, the resulting rule list is ordered. GRG performs several simplifications of the output rule list, for instance by merging rules related to the same output class or by deleting rules having no training samples satisfying their conditions.

Method by Lehmann et al. Lehmann *et al.* [104] proposed a pedagogical method applicable to BB classifiers working with binary inputs. The main idea of the algorithm is to use the underlying BB to classify all the possible input combinations, creating a rule for each input pattern. Then, the rule set is simplified and minimised by applying several strategies. The output of this procedure is a logic program, however each clause can be seen as a conjunctive rule. The complete explanation of this method together with all the formal demonstrations can be found in [104].

MGA MGA[42] [210] is a pedagogical algorithm designed for BB classifiers working on binary input variables. It outputs a list of conjunctive rules extracted by exploiting an evolutionary algorithm. The procedure performs two steps: a rule filtering phase to eliminate misleading rules is executed by taking into account measures of support, confidence and lift, followed by a rule set optimisation phase considering fidelity, coverage and complexity measures.

[40]BUR stands for Boost Unordered Rule learner
[41]GRG stands for Greedy Rule Generation
[42]MGA stands for Multi-objective Genetic Algorithm

HYPINV HYPINV[43] [149] is a pedagogical algorithm relying on the ANN inversion technique, i.e. finding the inputs which produce the desired output. It can handle both discrete and continuous input attributes and outputs classification rule lists in the form of conjunction and disjunction of hyperplanes. The authors suggest several optimisations for the produced rules—i.e. to reduce the rule application range, the number of rules and the number of attributes in each rule. The technique is classified as pedagogical because its output rules are not directly related to the ANN architecture or weights, even though the inversion procedure takes into account some ANN internal details.

Method by Yuan and Zhuang Yuan and Zhuang [212] proposed a pedagogical method to extract lists of fuzzy rules from BB models. It can be applied within classification tasks described by discrete input features. The starting rule set is created by applying the fuzzy logic properties to the training set instances. Then, a genetic algorithm is exploited to sequentially optimise the extracted rules w.r.t. classification accuracy, data set coverage and fitness. Fitness metrics takes into account the competitive contribution of a single rules w.r.t. the rule population.

Method by Hong and Lee, MDTF and MMFF Hong and Lee [86] proposed a pedagogical method to extract fuzzy rule lists from BB classifiers working with continuous input features. The goal of the algorithm is to automatically generate membership functions and classification rules. The procedure is composed of the following steps: *(i)* cluster and fuzzify the output data; *(ii)* build initial (triangular) membership functions for the input attributes; *(iii)* construct the initial decision table, according to the membership functions built in the previous step; *(iv)* simplify the decision table by removing redundant or unnecessary cells; *(v)* rebuild the membership functions during the decision table simplification, according to the executed simplifications; *(vi)* derive decision rules from the decision table.

Hong and Chen successively proposed MDTF[44] [84], an extension of the algorithm by Hong and Lee overcoming the complex decision tables and membership functions obtained with the original algorithm. Basically, MDTF first finds relevant input attributes and then groups these attributes' values into appropriate initial intervals.

The same authors then proposed MMFF[45] [85], an extension of MDTF that simplifies the intervals and the membership functions of each attribute before forming the decision table. MMFF decision tables are more concise and easier to process than those formed by MDTF.

Method by Ishibuchi et al. Ishibuchi *et al.* [94] proposed a pedagogical approach to extract lists of fuzzy rules from BB classifiers working with continuous input attributes. The main idea of the algorithm is to compose the conditional parts of the rules with the training instances' input attributes and the consequent parts of of the rules by exploiting

[43]HYPINV stands for extracting HYPerplanes using INVersion
[44]MDTF stands for Merging Decision Table First
[45]MMFF stands for Merging Membership Functions First

the underlying BB output predictions. A genetic algorithm is adopted to select a small number of significant output rules, since the procedure may extract a large number of them. This optimisation step focuses on the minimisation of the number of rules and the maximisation of the rule set predictive capability.

Methods based on artificial ant colony systems In the first decade of 2000, several knowledge-extraction procedures to pedagogically obtain lists of conjunctive classification rules by exploiting artificial ant colony systems have been developed. They accept discrete input attributes.

Ant-Miner [135] is the first proposed method of this category. Output rules are incrementally constructed and modified by agents inspired to ants. Each ant starts with an empty rule and iteratively adds to it one term at a time. Current partial rules correspond to current partial paths followed by the ants. Choosing a term to be added to a partial rule corresponds to the choice of a direction among all possible directions in the path of an ant. A pruning procedure is adopted to reduce the number of rule terms.

Liu *et al.* successively proposed two extensions of the Ant-Miner algorithm—namely, Ant-Miner2 [109] to reduce its computational cost and Ant-Miner3 [110] to improve its classification accuracy.

Finally, Martens *et al.* proposed Ant-Miner+ [120], a further extension of these algorithms achieving even higher accuracy. The authors introduce several modification to the original procedure, including *(i)* the use of directed acyclic graph to describe the ant environment, *(ii)* the discrimination between nominal and ordinal attributes to produce rule terms in the form of intervals, and *(iii)* the adoption of an early stopping strategy.

Extracting decision trees and tables from BB classifiers The following techniques are explicitly designed for BB predictors tackling classification problems. They produce as output rule trees or decision tables.

ID3, C4.5 and ID2-of-3 ID3[46] [140] is a pedagogical algorithm can be adopted to extract knowledge from a BB as a tree of classification conjunctive rules. The tree growth exploit the concept of *information gain* and it can only operate on discrete attributes.

The same author successively proposed C4.5 [142], an extension of ID3 aimed at overcoming its major drawbacks. C4.5 is capable to handle both discrete and continuous input data, as well as missing data. Furthermore, it has better accuracy when applied to data set with discrete input features that can assume a large number of different values. Finally, a pruning phase after the C4.5 tree creation is performed to reduce the number of extracted rules.

ID2-of-3 [128] is an extraction procedure similar to ID3 in its applicability and internal functioning. The main difference resides in its capability to extract *M-of-N* rules.

TREPAN and its generalisations TREPAN [53] is a pedagogical extraction method designed by the same authors of REAL. As this latter procedure, TREPAN is able to

[46]ID3 stands for Iterative DiChaudomiser, version 3

extract both *M-of-N* or conjunctive rules and it can only operate with binary input features, but it produces a tree of decision rules instead of a list. It is based on the induction of a decision tree constructed by following a *best-first* approach: the authors consider the best split among the possible ones w.r.t. the number of examples that it can classify as well as the predictive accuracy of the corresponding rule. TREPAN adopts two different stopping criteria: it stops the tree expansion when *(i)* the amount of internal nodes exceeds a user-defined threshold or *(ii)* the remaining internal nodes are converted into leaves since they have high probability to cover only data set instances belonging to a single class.

Browne *et al.* [26] and Torres and Rocco [180] proposed generalisations of the TREPAN algorithm capable of handling continuous and discrete input attributes.

Method by Krishnan et al. Krishnan *et al.* [101] proposed a method to pedagogically extract classification rule trees from underlying BB accepting continuous input features. The core idea of their method is to extract decision trees from data sets obtained by merging the original data set inputs and the underlying BB output predictions. The algorithm can be summarised as follows: *(i)* find the *class prototypes* (inputs correctly classified by the underlying BB); *(ii)* select the best prototypes and use them to induce the decision tree. The prototype generation exploits a genetic algorithm. The authors underline that having a large number of prototypes induces several advantages—e.g., robustness to noise, stability with outliers, and better generalisation capabilities.

DecText DecText[47] [23] is a pedagogical method to extract knowledge from BB classifiers in the form of decision trees. The authors developed several utility algorithms for DecText, so it differs from classical DT for the following reasons: *(i)* it adopts a peculiar splitting technique; *(ii)* it accepts continuous (and discrete) features, since it performs an *ad hoc* discretisation step; *(iii)* it exploit a customised pruning technique to enhance the fidelity of its output rules. The goal of DecText is to maximise the rule fidelity rather than accuracy. This implies that its output rules focus on mimicking the underlying BB behaviour rather than to be more accurate/general than the BB predictions.

SVM_DT SVM_DT [83] is an extraction method aimed at combining the generalisation ability of SVM with the comprehensibility of DT. The algorithm trains a SVM, then uses its results to generate a new data set to be used to train a decision tree classifier. The new data set is composed of the training instances correctly classified by the underlying SVM. SVM_DT can accept discrete and continuous input features and outputs a list of conjunctive rules extracted from the trained DT. We believe that, since rules are extracted from a DT, it is possible to introduce minor changes to the final algorithm step in order to obtain a tree of rules instead of a list.

[47]DecText stands for Decision Tree extractor

KDRuleEx KDRuleEx [158] is a pedagogical technique to extract decision tables from BB classifiers. It accepts continuous and discrete input data, however it applies a discretisation of the continuous attributes. The algorithm requires two user-defined parameters representing the minimum number of training examples to consider for the rule creation and a stopping criterion—e.g., rule length, accuracy level, etc. If there are not enough instances to satisfy the user-defined constraints, KDRuleEx generates random samples by using genetic techniques. The authors underline that KDRuleEx is not a recursive algorithm, so it is able to keep limited the size of stack and heap.

Extracting rule lists from BB regressors The following techniques are explicitly designed for BB predictors tackling regression problems. They produce output rule lists.

ITER ITER [91] is a pedagogical algorithm suitable for BB regressors. It only accepts continuous input features and produces as output a list of conjunctive *if-then* rules. The core concept of this method is the creation of hypercubes in the input feature space, where each hypercube is converted to an output rule and groups data set samples with similar output values. ITER can only associate a constant output value to each rule, hence it introduces an undesired discretisation in the predicted values that could hinder the predictive performance of the extracted rule set. The application of this technique may be critical when dealing with high-dimensional data sets in terms of computational time and rule readability, since the hypercube creation and expansion inside the input space present some criticalities, e.g., the hypercube expansion toward negligible input regions and a possibly very large amount of resulting hypercubes—and thus of output rules.

GridEx GridEx[48] [150] is another pedagogical procedure applicable to BB models performing regression tasks. As ITER, it only accepts continuous input features, it is based on the creation of hypercubes in the input feature space and it produces lists of rules with constant output values, however GridEx adopts a completely different inner logic. GridEx iteratively partitions the input space taking into account the importance of each input feature and the predictive performance of each hypercube. In other words, it creates more partitions for the most important features and avoid the further fragmentation of enough accurate hypercubes. To extract more robust rules, GridEx is able to produce a user-defined number of random samples inside each hypercube, using the underlying BB to predict the output values.

Extracting rules with general-purpose procedures The following procedures can be applied to BB designed for classification tasks as well as those dedicated to regression tasks.

[48]GridEx stands for Grid Extractor

CART CART[49] [25] is an algorithm for building decision trees that can be used for both classification and regression tasks. It accepts both continuous and discrete features. It is not a proper knowledge-extraction technique, but from its output tree it is straightforward to obtain a rule tree, since each path from the root to a leaf of the output tree is a single complete classification or regression conjunctive rule. The algorithm can be summarised as follows: *(i)* initialise the tree root node; *(ii)* find optimal splits and add new internal nodes and leaves accordingly; *(iii)* stop the algorithm based on one or more criteria—e.g., leaf number or tree depth. Pruning algorithms can be applied to reduce the number of leaves.

ANN-DT ANN-DT[50] [156] is a pedagogical knowledge-extraction methods that makes no assumptions regarding the inputs and outputs data it works upon. This implies that the method can be exploited in both classification and regression tasks and in presence of both continuous and discrete input features. The algorithm uses a sampling technique to build a decision tree, and it ensures that the sampled points are restricted only to the input space regions on which the underlying model is based. The tree construction takes into account the underlying BB responses to conduct a sensitivity or significance analysis of the different input attributes. Several stopping criterion can be adopted—e.g., stop splitting nodes when they have zero variance. ANN-DT adopts the same pruning method of CART.

SLAVE SLAVE[51] [31] is a pedagogical extraction algorithm suitable for BB classifiers as well as regressors. Its inputs can be continuous or discrete features and its outputs are fuzzy rule lists. SLAVE is an iterative algorithm. Each iteration can be summarised as follows: *(i)* exploit a genetic algorithm to learn one fuzzy rule describing the relationship between an output class and one of its input patterns; *(ii)* add the rule to the output rule set; *(iii)* penalise the rule, to let the algorithm learn different rules (e.g., remove from the training set the samples covered by the rule); *(iv)* return the rule set if it adequately represents the training instances, otherwise do another iteration. SLAVE is focused on the enhancement of the output rule simplicity and comprehensibility. Simplicity is considered w.r.t. variables (number of rule conditions) and values (determined by evaluating the distribution of the values assigned to relevant variables).

G-REX G-REX[52] [99] is a general-purpose procedure that can operate with any kind of input and output data—i.e., continuous or discrete values. G-REX is based on genetic programming and uses S-expressions to represent the individuals. It is similar to the GEX algorithm, but overcomes its limitations: G-REX is not limited to classification tasks and its rule are both exclusive and exhaustive. The basic shape of its extracted knowledge is a rule tree—where each node is associated to a single condition on a single

[49]CART stands for Classification And Regression Trees
[50]ANN-DT stands for Artificial Neural-Network Decision Tree algorithm
[51]SLAVE stands for Structural Learning Algorithm in Vague Environment
[52]G-REX stands for Genetic-Rule EXtraction

variable as well as to a disjunction or conjunction of conditions on more variables. Other output tree representations are available: interesting examples are constituted by the fuzzy tree and the hybrid linear regression models.

5.2 SKI Methods

We here provide a (non-exhaustive) list of SKI mechanisms. Information regarding such techniques are summarized in Table 4, where methods are organized according to the taxonomy presented in Section 4.2. A concise, yet exhaustive, description of each algorithm is then given below, analysing also their assets and liabilities. Detailed descriptions of such approaches are organized based on the *integration approach* they leverage (see Section 4.2.1).

Table 4: Knowledge injection algorithms.

Mechanism	Ref.	Year	Input			Integration			Aim	
			Rules	Graphs	Expert	Embed	Learn	Structure	Man.	Enrich
Ballard	[12]	1986	●					●	●	
KBANN	[183]	1990	●					●		●
Pinkus	[138]	1991	●					●	●	
Tresp et al.	[186]	1992	●				●	●	●	●
Giles et al.	[72]	1993			●			●	●	●
Cascade ARTMAP	[175]	1997	●					●		●
C-IL^{2}P	[56]	1999	●					●	●	
FOCA	[16]	2001	●				●	●	●	
Garcez et al.	[55]	2004	●					●	●	
Bader et al.	[8]	2005	●					●	●	
CODL	[36]	2007	●			●				●
Bader et al.	[9]	2008			●		●			●
Mintz et al.	[124]	2009		●		●			●	
RESCAL	[132]	2011		●		●			●	
PSL	[96]	2012	●					●	●	
Pools of Neural Binders	[137]	2013	●					●	●	
TransE	[22]	2013		●		●	●		●	
CILP++	[64]	2014	●					●	●	
TransH	[199]	2014		●		●	●		●	
TATEC	[71]	2014		●		●	●		●	
Bian et al.	[20]	2014			●	●	●			●
TRESCAL	[34]	2014		●		●			●	
Wang et al.	[197]	2015		●		●			●	
INS	[200]	2015		●		●			●	
LLE	[145]	2015		●		●	●		●	
Memory Networks	[201]	2015			●		●	●		●
TransR	[107]	2015		●		●	●		●	
DistMult	[209]	2015		●		●	●		●	
Pheno	[38]	2015		●			●			●
DLN	[184]	2016	●					●		●
Hu et al.	[89]	2016	●					●		●
KALE	[79]	2016	●	●		●	●		●	
HolE	[131]	2016		●		●	●		●	
ComplEx	[187]	2016		●		●	●		●	
LRI	[57]	2016		●			●		●	
Hu et al. (2)	[90]	2016	●					●		●
Mrksic et al.	[127]	2016		●			●			●
GCN	[98]	2017		●		●	●		●	

Continued on next page

Table 4: Continued.

Mechanism	Ref.	Year	Input			Integration			Aim	
			Rules	Graphs	Expert	Embed	Learn	Structure	Man.	Enrich
NTP	[144]	2017	•				•	•	•	
Analogy	[111]	2017		•		•	•		•	
Stewart et al.	[171]	2017	•				•			•
EqNet	[2]	2017			•		•			•
SBR	[59]	2017	•				•		•	
Diligenti et al.	[60]	2017	•				•		•	
GSNN	[115]	2017		•		•				•
NPE	[35]	2017			•	•				•
GRAM	[41]	2017		•		•				•
Fang et al.	[63]	2017		•			•			•
SLF	[208]	2018			•		•			•
∂ILP	[62]	2018	•				•	•	•	
LRNN	[170]	2018	•					•	•	
GAT	[194]	2018		•		•	•		•	
Multi-Omic	[112]	2018		•			•			•
CCM	[217]	2018		•		•				•
SGR	[106]	2018		•				•		•
ER	[73]	2018		•		•	•			•
LYRICS	[118]	2019	•				•			•
OSCAR	[76]	2019		•			•			•
RotatE	[172]	2019		•		•	•		•	
ERNIE	[216]	2019		•			•			•
KnowBERT	[136]	2019		•		•	•			•
LTN	[11]	2020	•					•	•	•
HAKE	[215]	2020		•		•	•		•	
Rule-INF	[211]	2021	•					•		•
DeepProbLog	[114]	2021	•					•	•	•

5.2.1 Techniques based on Model Structuring

In this section we present in details the list of SKI techniques that rely on network structuring, proposing to map the prior symbolic information at hand into structures of sub-symbolic models.

Method by Ballard Ballard presents one of the first works aiming at converting logical knowledge into NN structures [12]. Author formulates the theorem proving problem as equivalent to an energy minimization problem over structured networks built from first-order logic constraints. Each node in the structured network is associated to a state, while arcs are associated with weight of constraints, and constraint thresholds are mapped into states of the network. Therefore, Ballard shows that first-order logic constraints can be mapped in two coupled networks: *(i)* a network representing clause syntax; *(ii)* a network representing relationships between terms in different clauses. Finally, such structured networks are used for inferring logical knowledge via subsequent optimisation. The proposed approach accepts first-order logic constraints only, and is characterized by very trivial inferences, as it is one of the first studied on this field.

KBANN Authors propose to map the information contained in a prolog-like knowledge base into the structure of an Artificial Neural Network (ANN) [183]. The presented framework imposes the knowledge base at hand to be non-recursive and variable-free. The ANN is built defining one neuron for each entity in the KB, while constructive relations between such entities are mapped into the ANN edges. Additional units are added to the ANN and used to construct connections corresponding to disjunctive relations in the KB. The framework is tested on the task of recognizing biological concepts in DNA sequences, showing increased performances over traditional ANNs. However, the proposed approach seems to handle efficiently binary relations only.

Method by Pinkus Author considers building symmetric networks from a KB for query proving [138]. Given a propositional first-order-logic KB and a query over such KB, a symmetric network is constructed to match the KB and the given query. Pinkus proved that if a proof of the query exists, then it is represented by the global minima of the energy minimization problem linked with the constructed symmetric network. To reduce the complexity of such searching problem, an upper bound on the length of the query proof is imposed. However, the global minimum of the energy minimization problem gives proof only of the existence of such a solution for the given query. Further techniques are required for solution extraction from the global minima, and the author suggests leveraging the state of visible units to identify such solution.

Method by Giles et al. Authors propose the use of Recurrent Neural Networks (RNNs) in combination with predefined grammatical rules for deterministic finite-state automata transitions [72]. Such rules can be considered as expert knowledge rules, and are inserted in the RNN via manipulation of its structure. More in details, the knowledge integration approach is similar to KBANN [183], and relies on RNN weight initialization. A subset of the RNN weights are set to either $+H$ or $-H$, where H represents the rule strength H and is an arbitrary – expert-based – value. The proposed method allows for rules extraction as well, via clusterization of the space of RNN state neurons. Finally, authors also manage to show that wrongly defined prior knowledge, which was injected into the RNN, can be refined via RNN training.

Method by Tresp et al. A technique to embed expert knowledge into the structure of neural networks is proposed in [186]. Rule-based knowledge is used to prestructure a neural network, initializing its weights and connections. The NN is built using basis functions characterized by continuous positive values which represent the certainty of given rules. Outputs of such basis functions are then weighted together to obtain the output of the NN, representing the composition of activated rules. Moreover, to preserve logical knowledge during training, different penalty terms are added to the loss function. These penalties are designed to penalize terms deviating from their initial value, aiming at preserving the starting logical knowledge. Therefore, this method leverage a combination of *model structuring* and *guided learning*. Finally, authors propose a knowledge extraction approach as well. After NN training, logical rules can be extracted and refined from such NN, to analyse their modifications.

C-IL^{2}P The Connectionist Inductive Learning and Logic Programming (C-IL^{2}P) system is presented [56], aiming at integrating inductive learning with deductive learning. Technically speaking, the proposed framework tackles inductive learning using examples and background knowledge, while integrating deductive learning from Logic Programming. Background knowledge under the form of logic program is translated into a NN structure which can be trained over a set of examples, introducing induction. Injection of KB into the NN is obtained through simple architectural approach: each general clause of the grounded logic program is mapped from the input layer to the output layer of the NN through one neuron in the single hidden layer of the NN. Therefore, such approach support shallow – one hidden layer – NNs only. Neurons corresponding to positive (negated) clauses are connected to the hidden layer neuron using weight W $(-W)$ respectively, while hidden layer neurons are connected to the output layer neuron with weight W. Moreover, input and output neurons that share the same label are recursively connected from the output to the input of the network with weight 1, so that output values can be propagated back to the input in the next rule chaining computation.

Cascade ARTMAP Cascade Adaptive Resonance Theory Mapping (ARTMAP) is introduced as a novel technique for integrating if-then rules into neural networks [175]. This model represents a generalization of fuzzy ATMAP [30], in which authors propose a novel rule insertion algorithm which translates symbolic knowledge into a cascade architecture for neural networks. More in details, the mechanism identifies a symbol table where attributes of the symbolic knowledge at hand are translated into positive or negative samples to be used as training patterns. As such, the proposed system is capable of representing intermediate attributes and rule chains that would be complex to integrate otherwise. Chain rule firing is considered during the training of the obtained NN, introducing a backtracking algorithm, aiming to identify all rules responsible for the firing of a specific prediction. Responsible components are then optimised depending on the correctness of the aforementioned prediction, leveraging a mini-match tracking procedure if necessary to avoid blame assignment errors. It is here important to notice that this learning procedure is match-based, not error-based, avoiding wash-away issues of the knowledge represented by the learnt parameters of NN units. Finally, the proposed cascade ARTMAP method can be translated into a set of symbolic rules using a generalized rule extraction procedure. More in details, authors propose to select a small set of rules from the obtained networks based on their confidence factors. Therefore, this mechanism constructs a rule-insertion (NN construction), rule-refinement (training), and rule-extraction cycle.

FOCA A First-Order extension of the Cascade ARTMAP system (FOCA) is proposed [16]. The proposed system aims at solving the first-order-logic theory refinement problem using neural networks. Authors builds on to of Cascade ARTMAP [175], which is a knowledge-based neural network used for propositional theory refinement, and extends it to support first-order-logic knowledge. Such extension involves mapping first-order-logic to the structure of the NN. Each neuron in the NN represents a predicate symbol of the problem, while neuron activation represents the tuples of terms with whom such predicate is involved. ARTMAP choice, match functions, and learning algorithm are also extended to support fisrt-order-logic knowledge. The techniques leveraged, allow FOCA to learn first-order theories, similarly to an Inductive Logic Programming system.

Method by Garcez et al. The concept of *fibring* Neural Networks is introduced, aiming at combining logical symbolic knowledge with connectionist approaches typical of NNs [55]. The fundamental idea behind fibred NNs is to compose networks not only using interconnected neurons but also interconnecting other networks, basically forming recursive ensembled architectures. Therefore, fibred NNs are capable of implementing recursion, enabling them to perform symbolic computation, and giving them a Neural-Symbolic characterisation. Networks interconnections require the introduction of so called *fibring functions*, which define how the recursive architecture must behave by defining how the NNs in the ensemble relate to each other. As an example, a fibring function may allow the activation of neurons in one network (A) to influence the change of weights in another network (B). Authors then show how fibred NNs are able to approximate any polynomial function to any desired degree of accuracy, thus being more expressive than standard feedforward networks.

Method by Bader et al. Authors in [8] propose to represent first-order logic programs as fibred neural networks, building on top of Fibring Neural Networks [55]. Fibring functions are associated with NN neurons which map some weights w of the network to new weights w', depending on w and the input of the neuron at hand. Fibring functions in these approach are used to inject symbolic knowledge, under the form of acyclic first-order-logic programs, into the structure and weights of the NN at hand. In order to fiber logical knowledge into NNs, authors deploy a global counter n, which fibres the kernel and evaluates whether the atom of level n is contained in the single-step operator or not. Authors then present few fibring function implementations for specific logical scenarios—e.g., gating, remainder, powers. Following the proposed approach, fibred NNs are capable of approximating the single-step operator of the logic program, capturing its semantics.

PSL Probabilistic Soft Logic (PSL) is presented as a framework for collective, probabilistic reasoning in relational domains [96]. PSL is defined over first-order logic rules. These rules are then modelled as a template for Markovian models, which leverage random variables with soft truth values from the interval $[0, 1]$. Having constructed the Markovian model, inference can be modelled as a continuous optimisation task, which can be solved efficiently.

Pools of Neural Binders Authors present a paradigm to encode and process relational knowledge into Boltzman machines [137]. A working memory is added to the structure of the boltzman machine, along with a long-term synaptic-memory which is capable of storing the knowledge base at hand. Whenever a query over the knowledge base at hand is requested, a compact variable binding mechanism is activated to dinamically allocate neurons into ensembles of neurons. Such ensembles are in charge of iteratively retrieving an item from the knowledge base, up untill the solution for the required query is found in the working memory. Authors propose to encode relational graphs as neural activations available in the working memory and to use constraints encoded in synapses to retrieve and process relational structures. Procedural and declarative knowledge are then expressed as constraints and used to generate weighted connections between neurons.

CILP++ A method for Inductive Logic Programming (ILP) which leverages NNs – called *CILP++* – is proposed [64]. CILP++ builds on top of the similar CIL^{2}P framework and extends it with the introduction of Bottom Clauses, obtained by means of propositionalization. Bottom clauses are used to bound the ILP hypothesis search space. Bottom clauses are mapped directly onto numerical vectors, which are then embedded onto the NN structure following the CIL^{2}P approach. Starting from a Background Knowledge (BK), the NN is created initializing a hidden layer neuron for each clause in the BK. Then, each body literal in BK is associated with an input neuron, and each head literal in BK is associated with an output neuron. Neurons corresponding to positive (negated) literals are connected to the hidden neuron using weight W $(-W)$ respectively, while hidden neurons are connected to the output neuron with weight W. Moreover, input and output neurons that share the same label are recursively connected from the output to the input of the network with weight 1, so that output values can be propagated back to the input in the next calculation of rule chaining.

Memory Networks MemNNs are presented as a novel paradigm for building Neural Networks [201]. Authors aim at tackling the memory lacking problem of NNs – more in details RNNs – which affects models dealing with text processing and question answering. The framework proposes to insert a novel memory model in the construction of the NN, which can be used to store information – i.e., embeddings – of previously seen words or sentences. The memory module can then be used and updated by successive modules of the NN structure, in order to keep relevant information only and store past information that other RNNs would forget. The memory module works with word and sentences embeddings for ease of integration with other standard modules of NNs. The proposed module is then trained jointly with others NN modules, thanks to custom loss function definition which takes into account the memory storage for past embeddings.

DLN Authors propose Deep Logic Networks (DLN), aiming at extracting and injecting logical rules into NNs [184]. Logic rules are presented as if-and-only-if predicates which are associated with a numerical confidence value. Such rules can be hierarchically ordered, allowing the selection of rules to keep. Given a set of hierarchical confidence rules, the knowledge of those rules can be encoded in the structure of a Deep Belief

Network (DBN). Knowledge is injected adding units to hidden layers of the DBN, corresponding to the heads of confidence rules. These units are then connected to the inputs that corresponds to the literals present in the body of the rule, and the weight assigned to this connection is the confidence value of the corresponding rule. Hierarchical rules are then encoded in different layers of the DBN whose depth depends on the hierarchy level. Following this approach prior knowledge is used to partially fix some upward connections in the network and guide the learning procedure. The proposed knowledge insertion approach was tested on the DNA Promoter Dataset and on the MNIST data set, showing increased performance over scratch NNs.

Method by Hu et al. An iterative distillation method is proposed to inject first-order grounded logical rules into NN weights [89]. The proposed framework leverages an iterative rule knowledge distillation procedure, transferring structured information of logic rules into NN parameters. More in details, at each training iteration a rule-regularized teacher NN is constructed, and a student NN is trained to imitate the predictions of the teacher. The teacher NN is constructed at each training iteration via projection of the student NN to a rule-regularized subspace defined by a set of first-order logic grounded rules. The projection is obtained minimizing the Kullback-Leibler divergence between the prediction distribution of the NN and the rule-regularized prediction distribution. The framework was tested on CNN for sentiment analysis, and RNN for named entity recognition, showing improvements over previous systems.

Method by Hu et al. (2) Authors build on top of their previous knowledge distillation approach [89], generalizing the one-sided distillation method to a mutual distillation framework. Here authors propose to mutually transfer information between the NN and the logic constraints at hand [90]. Distillation is used to inject first-order grounded logical rules into NN weights via projection of the NN to a rule-regularized subspace. Identically to their previous approach, the projection is obtained minimizing the Kullback-Leibler divergence between the prediction distribution of the NN and the rule-regularized prediction distribution. However, authors propose, not only to distill domain knowledge into the NN, but to extract and refine domain knowledge from the NN, depending on its prediction. NN distillation and knowledge rules refinement are then jointly trained using a mutual distillation loss function during training.

NTP Authors introduce Neural Theorem Prover (NTP) [144]. Neural Networks are built to solve logical queries over a specified knowledge base, available under logic program form. The Prolog's backward chaining algorithm is used to build recursively the structure of the NN. Specifically, differentiable *unification*, *or*, and *and* modules are defined and used to build the NN and treat vector representations of symbols. Positive and negative queries are sampled from the KB at hand and used to train the corresponding NN. Training is obtained using a custom-made Neural Link Prediction auxiliary loss term. The obtained NN learns to represent similar logical concepts using vectors close to each other in the obtained sub-symbolical vectorial space. Moreover, the constructed

NN can be used to prove queries via computation of the gradient of proof successes with respect to vector representations of symbols

∂ILP Authors introduce ∂ILP [62], a differentiable NN framework for Inductive Logic Programming (ILP), reliable against noisy data. An ILP problem is translated into a binary classification problem, where discrete operators are replaced by their differentiable counterparts. Therefore, ∂ILP defines a differentiable deduction structure – Neural Network – working over continuous values. Practically, ∂ILP constructs a NN whose structure reflects the grounded version of the background knowledge at hand. Meanwhile, the gradient – computed over such architecture and used for the classification loss minimization – implements a continuous form of induction. Therefore, the proposed architecture is capable of solving ILP problems via differentiable optimisation. Authors test the proposed framework over a set of different tasks, showing its ability to learn moderately complex programs, invent novel useful predicates and recursive clauses.

LRNN Authors in [170] introduce Lifted Relational Neural Networks (LRNN), trying to combine first-order logic knowledge with NN computations. LRNN is inspired by traditional lifted frameworks where patterns for a general task are defined via specification of the problem structure. The general structure is then used to derive specific ground models for each example in the task to be solved. LRNN leverages lifted frameworks to specify grounded first-order logic rules that define the structural templates of NNs. LRNN utilizes such structural templates to build one NN for each training and testing example of the task to be solved. Importantly, the different NNs share the weights specified by the general structural templates, therefore injecting general knowledge to each NN. These shared weights can then be learned via standard backpropagation. Authors define simple structuring rules, assigning one neuron of the NN for each atom, fact, rule and aggregation available in the FOL knowledge base.

SGR Authors propose a novel NN layer called Symbolic Graph Reasoning (SGR) layer [106]. The proposed layer aims at performing reasoning over a group of symbolic nodes whose outputs explicitly represent different properties of each semantic in a prior knowledge graph. More in details, authors considered convolutional NNs dealing with images data and propose a general layer, which is compatible with other convolutional layers. The proposed layer is composed of three modules: *(i)* a local-to-semantic voting module which is in charge of converting local features into semantic graph; *(ii)* a graph reasoning module which elaborates the information belonging to the semantic graph, producing a new semantic graph as output; *(iii)* a semantic-to-local mapping module which maps back the semantic graph into local features of images. The proposed layer can be embedded into the structure of a NN and can be trained along with it, obtaining a knowledge enriched NN.

LTN Authors in [11] propose Logic Tensor Networks (LTN). LTN are computational models which supports learning and reasoning via embedding of logic into many-valued real tensors. Embedding of logic knowledge into tensors is reached thanks to a first-order logic called Real Logic. This peculiar form of logic is end-to-end differentiable – thanks to the implementation of differentiable operations equivalent to conjunction, disjunction, aggregation, etc. – and serves as a representation language for NNs. LTN converts Real Logic formulas – e.g., $\forall x(cat(x) \rightarrow \exists y(partOf(x, y) \wedge tail(y)))$ – into TensorFlow[53] computational graphs. Such formulas express prior knowledge to satisfy during learning, queries over the data or, more generally, any logic information to be leveraged by the NN. Intuitively, the NN built via LTN is capable of learning from labelled examples, while maintaining the logic imprint available in its computational graph. Therefore, the proposed neurosymbolic approach allows injection of symbolic knowledge in multiple tasks like classification, regression, clusterization, etc.

Rule-INF Authors in [211] propose a novel technique to extract and inject rules from Stacked-Denoising-AutoEncoders (SDAE). The rules are extracted in if-else form analysing and representing the structure and interconnections of the pretrained SDAE at hand. Extracted symbolic rules seem to be suitable for the representation of quantitative reasoning in a NN. The extracted rules are then used to build a new SDAE model, focusing particularly on the NN weights initialization. Connections between hidden layer neurons and the input neurons of the SDAE are determined from the extracted rules rule. The weights between hidden and input layers are set to ± 1 based on the validity of each rule. The obtained novel SDAE is then trained again, ignoring those weights initialized via logical rules. The proposed framework is tested on various benchmark data sets, showing that rule insertion in SDAEs allows boosting performance of feature learning.

DeepProbLog Authors introduce *DeepProbLog* [114], which is a novel framework to integrate neural networks and probabilistic logic programming. The proposed framework is end-to-end differentiable, and aims at optimising both sub-symbolic representation and probabilistic logic. The framework was obtained extending the ProbLog language via introduction of neural predicates—i.e., predicates whose probability estimates are accomplished by learnable NN models. Therefore, DeepProbLog programs are translated into tensorial computational graph, whose structure reflects the rules contained into the DeepProbLog program. During end-to-end training, ProbLog knowledge base is grounded and converted into real-valued vectors, which are embedded into the differentiable deep-learning pipeline. aProbLog [97] is then used to compute the gradient of the loss, allowing optimisation of both the probabilistic logic program and the neural networks parameters.

5.2.2 Techniques based on Guided Learning

In this section we present in details the list of SKI techniques that rely on guided learning, proposing to guide the learning procedure of of sub-symbolic models. These techniques

[53]`https://www.tensorflow.org`

propose to translate symbolic information into constraints or components of the loss function to be used during model training.

CODL Authors propose the COnstrained Driven Learning (CODL) approach [36]. Domain knowledge is embedded into learning procedure of sub-symbolic systems via the use of constraints set. More in details, the domain knowledge at hand is converted into a set of constraints, defined as functions that indicates whether the input/output sequence violates KB properties. The constraints are then injected in the semi-supervised learning procedure of an HMM model, introducing a distance function term between the given output and the space of outputs that respect the KB constraints.

Method by Bader et al. Authors propose to guide the backpropagation algorithm, via successive training iterations [9]. Subsequent training iterations are corrected embedding correction rules into the NN. These correction rules are obtained analysing the error of the NN. Practically speaking, authors propose a training procedure which is composed of the following steps: 1) initialise the network; 2) repeat until stopping condition: 2.a) train network for n cycles; 2.b) analyse NN errors and build correcting rules; 2.c) embed correction rules in NN. Authors then propose a proof of concept training, where rules are extracted and embedded using hand-crafted criteria. Therefore, general approaches for rule extraction from errors and rule embedding are still missing for generalization of this approach.

Method by Bian et al. Authors present an empirical study on using morphological, syntactic, and semantic knowledge as helpers into a word embedding model [20]. The proposed frameworks aim at achieving high-quality word embeddings and study if and how predefined knowledge can help embedding procedures. Knowledge available under the form of syntactic, morphology and semantics is used as an additional input information for the supervision learning of bag-of-words embeddings. For each of the knowledge considered an auxiliary task is defined, which needs to be maximized during training along with the average log probability used to obtain the bag-of-words embeddings. Such auxiliary tasks serve as a guide for the training procedure, aiming to obtain knowledge-enriched embeddings. Depending on the task and knowledge at hand, different embeddings of the input knowledge are used, which always rely on knowledge vectorization.

Pheno The Deep Computational Phenotyping system (Pheno) is proposed, which aims at leveraging existing clinical domain knowledge available under the form of ontologies as a prior during training procedure of NNs [38]. In order to achieve this goal, authors propose a framework relying on the transformation of such ontology into a graph Laplacian. The graph laplacian is then used as a simple, computationally efficient regularizer for the training loss function. This procedure allows the NN to be guided in the training task and learn to learn more clinically relevant features.

LRI Authors present a Lifted Rule Injection (LRI) framework for incorporating implication rules into knowledge base inference tasks [57]. Implication rules are mined from WordNet and considered well-established. The proposed framework aims at surpassing previous limitations avoiding propositionalization of first-order logic rules. Indeed, propositionalization does not scale well with large knowledge bases. Therefore, authors propose to leverage partial ordering among embeddings of entity-relation tuples. Entity-relation tuples are mapped into an approximately boolean space, while the partial ordering among them is imposed via lifted loss formulation. More in details, given an implication rule $r_p \Rightarrow r_q$, it can be imposed by requiring that every entity-relation tuple is at least as compatible with relation r_p as it is with r_q.

Method by Mrksic et al. Authors present a novel counter-fitting method which aims at injecting the concepts of antonymy and synonymy into the word embedding procedure [127]. The final aim of the framework is to obtain word embeddings characterised by constraints that satisfy the antonymy and synonymy concepts. More in details, in order to enable synonyms and antonymous characterisation into word embeddings, constraints over the loss function are used. In particular, one antonym repel term is used to push antonymous words vectors away from each other, while a synonym attract term is used to bring vectors of synonymous word pairs closer together. Authors show that embedding these concepts into the vectorial representation space improves the capability of the embedding method to represent and judge semantical similarities. Synonyms and antonyms are considered to be available under the form of a knowledge graph, used as input of the framework.

Method by Stewart et al. Authors in [171] introduce a novel approach for training neural networks over a knowledge-base guided task such as physics-guided tasks—i.e., trajectory prediction, etc. The proposed approach aims at removing need for supervised learning of NNs, by specifying logical constraints – e.g., physics laws – that should hold over the output space. Therefore, the need for specification of input-output examples is dropped. However, such approach introduces the need for embedding such domain knowledge – e.g., physics laws – into appropriate loss function terms or constraints. Indeed, authors define few custom-made loss functions to be used for the presented experiments, while not giving general knowledge of how this approach could be generalized.

EqNet Authors propose Neural Equivalence Networks (EqNet) [2], aiming at tackling neural representation of algebraic and logical expressions. Semantic equivalence is used as the training task for EqNet, in order to learn continuous semantic representations of logical expressions. EqNet rely on the structure of Tree-based RNNs [174] to embed the syntactical level of sentences. However, they extend such framework to abstract away from the syntactical level only and tackle semantic equivalence. Indeed, semantic is linked to syntax, but may be also mislead by it. To abstract away from the Tree-based RNN representation, authors introduce a new layer of standard NNs and a custom training objective. The new NN layer combines syntactical information, while the custom training objective aims at injecting two properties into the embeddings: abstraction and reversibility.

SBR Authors propose a Semantic Based Regularization (SBR) framework for integrating first-order logic (FOL) rules into kernel machines such as SVMs [59]. The proposed framework define multi-layered structure having kernel machines at the input layer. The output of such layers are fed to higher levels which implement a fuzzy generalization of the FOL knowledge. FOL formulas are translated into constraints applied to the training loss function, using fuzzy translation techniques. Moreover, authors propose to leverage a collective classification method to enforce the constraints obtained from FOL rules also on the test set, thus exploiting the full expressiveness of FOL. FOL rules are translated to constraints by means of t-norm fuzzy logic, where the rules are associated with continuous values of truthfulness, and by means of quantifiers, meaning that all variables are quantified and all the quantifiers ($\forall$, $\exists$) are placed at the beginning of the expression.

Method by Diligenti et al. Authors propose a general approach to tackle the task of integrating prior knowledge – under the form of first-order logic rules – into NNs [60]. The proposed framework builds on top of the Semantic Based Regularization approach [59], and focuses on a specific and narrow scenario: each task to be learned by the NN corresponds to a predicate in the knowledge base. Therefore, the NN can be seen as a predictor of predicates in the KB at hand. To inject logic knowledge into the NN, the SBR approach is used. Prior knowledge is translated into a set of constraints which are integrated into the training loss function. Therefore, such logical knowledge is embedded in the NN during training via backpropagation.

Method by Fang et al. Authors propose to introduce knowledge graphs embedding into object detection mechanisms [63]. The proposed framework employs the notion of semantic consistency to quantify knowledge of similar objects and improve object detection. More in details, the predictions P of an existing object detection model are refined in order to satisfy knowledge constraints available in a predefined knowledge graph. The refined predictions $\hat{P}$ should therefore satisfy a specific knowledge base and are considered a knowledge-aware enhancement of P. More in details, authors propose two approaches to inject knowledge and refine predictions: *(i) concepts co-occurrence* can be measured in order to score less the independent concepts than dependent objects—here dependent onjects are considered as those objects that appear together in the same image; *(ii)* semantic consistency over knowledge graphs can be obtained via *random walks* over the knowledge graph at hand—therefore, objects linked by the same path will be considered as dependent and will be assigned a higher probability of appearing in the same image. Knowledge-refinement scores are then used for fine-tuning the predictions, introducing a custom-made term in the loss function of the object detection model. The novel loss term depends on the approach selected for prediction refinement and might be cumbersome.

SLF Authors propose the Semantic Loss Function (SLF), a loss function aiming at bridging the gap between NN and logical constraints [208]. The proposed loss function captures how far is the output of the NN to satisfy some constraints which should be fulfilled over the outputs. Constraints regarding the output structure are considered under various form—e.g., one-hot encoding, preference ranking, and path validity. Authors propose different handcrafted loss terms, depending on the form of the constraints to be imposed. The novel loss term is used in combination with standard loss function, to guide the NN learning satisfactory output structures. The proposed framework is applied to simple, yet significant learning tasks, showing promising performance when considering complex output structures like preference ranking.

Multi-Omic Authors propose a technique to embed proteins-interaction graph knowledge into an auto-encoder NN [112]. The auto-encoder NN is used to predict clinical outcome of a patient from multi-omic data—thus the name. The proposed approach leverages loss regularization via injection of knowledge under the form of the Laplacian matrix of the proteins-interaction graph. The regularization term based on the graph Laplacian matrix can enforce strongly connected variables to behave similarly in the model, while unconnected variables are free to contribute differently. This allows to integrate genetic pathway information into the auto-encoder model, which can then be trained end-to-end supervisedly or unsupervisedly.

ER Author propose an Explicit Retrofitting (ER) model [73]. Retrofitting is the process of fine-tuning word embeddings using external lexical knowledge. This process is usually implemented to achieve a better embedding of some semantic relations. Authors here propose to leverage external lexico-semantic relations and transform them into training examples used to learn ER. The proposed model aims at learning global specialization function in order to specialize the vectors of unobserved words. More in details, synonym and antonyms relations knowledge is mapped into distance scores between entities that should be matched by the embedding model. The fine-tuning procedure is then applied, trying to modify learnt word embeddings in order to match the distance scores between synonyms and antonyms.

LYRICS Authors present a general framework, called LYRICS [118], which aims at bridging logic reasoning and deep learning. The proposed framework leverage the translation of First-Order Logic (FOL) predicates and functions into real-valued tensor and Tensorflow computational graphs. The obtained computational graphs are then embedded in the computational graph of standard deep learning tools—e.g., NN. Meanwhile, the logical knowledge is shown to be injectable to NN systems via means of constraints for the optimisation task at hand. FOL formulas are converted into a set of real-valued loss constraints, which participate to the overall optimisation problem. LYRICS is implemented in TensorFlow and is architecture agnostic, meaning it allows integration with theoretically any NN model. The constraints are presented using the frontend of the framework, leveraging a FOL-based declarative language. Authors test the framework over various scenarios – e.g., supervised classification, semi-supervised classification –, showing improved performance over raw NN counterparts.

OSCAR Authors in [76] propose an Ontological Semantic Composition Augmented Regularization system—namely OSCAR. OSCAR is a pretraining regularization technique, which aims at injecting world knowledge and ontological relationships into deep neural networks. Authors state that in order to inject world and domain knowledge from a knowledge base into the NN, it is required to semantically decompose the information about an entity into the supporting information about its constituent words. To do so, OSCAR: *(i)* first detect entities in sentences; *(ii)* then the sequence of subwords – some entities may be composed of multiword phrases like *police officer* – corresponding to each entity are semantically composed to produce an entity-level encoding; *(iii)* and finally the average energy between the composed entity encoding and the pretrained entity encoding from the ontology is used as a regularization term in the pretraining loss function.

ERNIE Authors in [216] introduce ERNIE, a novel enhanced language representation model. The proposed model aims at leveraging and embedding both large-scale textual corpora and knowledge graphs (KGs) into the trained language representation model. ERNIE can take full advantage of lexical, syntactic, and knowledge information simultaneously. Indeed, entity mentioned in a text are firstly recognized and then aligned to their corresponding entities in the predefined KG, which represent prior knowledge. The graph structure is then encoded using well-known embedding algorithms like TransE [22], and the obtained embedding is used as input for ERNIE. ERNIE is trained following a masked language model – similar to BERT [58] –, where sentence completion is modelled as an entity selection problem from the KG at hand, to better fuse textual and knowledge features. Therefore, the prior knowledge available is injected in the NN via loss regulation and training manipulation.

KnowBERT Authors propose KnowBERT [136], a modified version of the BERT [58] model proposed for embedding words and sentences in the Natural Language Processing area. KnowBERT propose a modification of BERT to embed knowledge bases (KBs) into the model. The aim is to enhance the embeddings with structured, human-curated knowledge. Authors first use an entity linker module to retrieve relevant entity embeddings for entities belonging to the knowledge base at hand. Entity representations are then updated leveraging a word-to-entity attention module, and a modified custom-made loss function. Finally, the entity linker and the attention modules are trained jointly, and a supervised entity linking task is added to the traditional BERT training setup.

5.2.3 Techniques based on Knowledge Embedding

In this section we present in details the list of SKI techniques that propose to transform the unstructured symbolic information at hand into structured data – e.g., vectors, matrices, tensors, etc. –, allowing the sub-symbolic model to learn from them, therefore injecting the prior knowledge available.

Method by Mintz et al. A distant supervision approach is proposed for injecting relational knowledge – available under the form of knowledge graphs – into NNs [124]. The proposed system is tested on the content extraction task typical of Natural Language Processing (NLP), showing improved performance. The logical knowledge in a graph is used to automatically annotate texts, which is then used to train natural language processing systems. More in details, authors consider each sentence that matches related entities in a graph as a training sample. Therefore, relational knowledge is injected – indirectly – into NNs through selection of training data.

RESCAL RESCAL is proposed as a factorization approach for learning relations between entities [132]. The proposed approach takes into account inherent structure of relational data. Prior knowledge is considered to be available under the form of knowledge graphs, defining triplets of entities and relation between them. Relations are embedded into a three-way tensor via rank factorization, which can be solved finding the solution for a regularized minimization problem. RESCAL is a bilinear model which captures the interactions between the two entities using multiplication between their vectors. The results on various data sets as well as the runtime performance are very competitive and show the potential for tensor embedding of entities relations.

TransE A relationship embedding framework (TransE) is proposed, which aims at encoding relationships between entities of a knowledge base [22]. Authors propose to model relationships between entities by interpreting them as translations operating on the low-dimensional embeddings of the entities involved. Translation between entities embeddings is then practically implemented using simple distance function between entities. Authors suggest to leverage an energy-based framework, where the energy of a triplet is equal to $d(h + l, t)$, with h and t being two entities, l being the relationship among them, and d being some dissimilarity measure—either the L1 or the L2-norm. The energy of valid triplets – i.e., relationships that hold – is minimized, while it is maximized for corrupted triplets—i.e., false relationships. In order to optimise the energy of such triplets, a margin-based ranking criterion term is introduced in the loss function. Therefore, such approach mixes together the *knowledge embedding* and the *guided learning* approaches, showing improved performance over the relation prediction task.

TransH Authors propose TransH [199], a novel relations embedding model, which builds on top of the TransE [22] framework. The proposed framework aims at surpassing TransE limitation of considering relations as simple translations between entities. To obtain a good trade-off between model capacity and efficiency, TransH propose to model a relationship as a hyperplane together with a translation operation on it. Therefore, TransH introduces a more refined step of complexity, which aims at preserving some mapping properties of relations which should be satisfied in the embedding world and for which simple translations are not sufficient. Authors state that these properties are reflexivity, one-to-many, many-to-one, and many-to-many. Similarly to TransE, the embedding procedure for relationships between entities is trained via minimization of a custom margin-based ranking loss function that aims at discriminating between true relations and false relations.

TATEC A knowledge graph embedding mechanism is proposed – namely TATEC –, having the aim of identifying missing relationships in knowledge bases [71]. The embedding mechanism proposed here, relies on the scoring of triplets of facts, depending on bigram and trigram relations. Bigram scores ideally identify the proximity between the head and the tail of a triplet, identifying the similarity between entities. Trigram scores, on the other hand, aims at weighting directly the importance – i.e., truthfulness – of a triplet. Following this approach, entities closely related and truthful triplets are assigned with higher scores, obtaining a truthful embedding of the knowledge base at hand. A ranking criterion is used as training objective – via introduction of a custom loss term – in order to optimise the embedding procedure and obtain a significant embedding scheme.

TRESCAL Authors introduce a novel approach for embedding information belonging to a knowledge base into a relation extraction mechanism, namely TRESCAL [34]. The proposed relation extraction mechanism does not rely solely on textual information, as commonly done, but it relies on the integration of the knowledge base into the execution pipeline. More in details, authors propose a tensor decomposition approach for knowledge base embedding. The proposed method builds on top of RESCAL [132], introducing two technical innovations. First, authors decided to exclude the triples not satisfying the relational constraints. Secondly, they reduce the computational complexity of the approach introducing a novel mathematical technique which deals more easily with regularization terms.

Method by Wang et al. A novel approach to extend incomplete knowledge bases (KBs) is proposed [197]. The proposed framework leverage embedding techniques like RESCAL [132], TRESCAL, and TransE [22] to embed the knowledge base into a latent space and make inferences regarding missing information with the help of logical rules. The inference problem is then formulated as an Integer Linear Programming (ILP) problem where the objective function is generated from the embedding, while ILP constraints are obtained from predefined rules. The predefined rules used as constraints are the following: *(i)* facts that already exist are very likely, but not necessarily to be true; *(ii)* arguments of a relation should be entities of certain types; *(iii)* for 1-To-Many/-Many-To-1 relations, the first/second argument can take at most one entity, for 1-To-1 relations, both arguments can take at most one entity; *(iv)* simple implication. Following this approach predicted missing facts will be the most preferred by the embedding models, and will comply with all the rules imposed.

INS A novel approach for knowledge base embedding is proposed, which particularly suits the knowledge base completion task on large knowledge bases [200]. Authors propose an embedding-based model to learn distributed representations for both entities and relations in the existing knowledge base. The proposed embedding aims at putting closer to each other existing entities-relation triplets, while abandoning randomly generated entities-relation triplets. Learned representations of entities and relations are then used to calculate similarity scores and form a new smaller knowledge base. The newly obtained knowledge base encoding is then used to infer – using Markov Logic Networks – entities-relation triplets required for the knowledge completion task.

LLE Authors introduce a novel paradigm – namely Low-rank Logic Embeddings (LLE) – for learning low-dimensional embeddings of entity-pairs and relations [145]. The proposed approach aims at combining the advantages of matrix factorization with first-order logic domain knowledge. Given a binary matrix consisting of observed facts over entities and relations, matrix factorization and first-order logic formulae are used to learn k-dimensional relation and entity-pair embeddings, approximating the observed matrix—i.e., facts. Author propose two techniques for injecting logical background knowledge: *(i) pre-factorization inference*: performing logical inference on training data and add inferred facts as additional training data; *(ii) joint optimisation*: including a loss term for the logical formulae directly in the matrix factorization objective, which results in an optimisation of embeddings for factual training data and first-order logical background knowledge.

TransR TransR is proposed [107], which is a relations embedding model, which aims at surpassing some limitations of TransE [22] and TransH [199]. Previous Trans* models put both entities and relations within the same semantic space. This is indeed an oversimplification of the problem. An entity may have multiple aspects and various relations may focus on different aspects of entities. This makes it somehow insufficient to model all relations and entities over a single common space. TransR propose to overcome this issue, building entity and relation embeddings in separate spaces (entity space and relation spaces). Embeddings are then learnt by projecting entities from the entity space to the corresponding relation space. On the other hand, relations are modelled as translations operations between the projected entities in the relation space, similarly to what is done for TransE. Finally, similarly to TransE and TransH, the embedding operation is trained using a custom margin-based score function objective during training.

DistMult Authors present a general framework for embedding entities and relations available in a Knowledge Base [209]. The KB at hand is considered to be available under the form of entities-relations triplets. This work then presents a general procedure for obtaining a learnable embeddings of KBs and evaluate some possible specific modules on knowledge inference tasks. More in details, authors propose to generalize previously available embedding models – like TransE [22] – to a unified learning framework. In this framework entities are represented as low-dimensional vectors learned from a NN and relations are mapping functions between these vectors. Authors consider to leverage bilinear and/or linear mapping functions and showed that simple formulation of bilinear model can outperform available embedding models. The proposed framework can also be leveraged to extract logical rules from KB at hand. Learning of such embedding of KBs is then modelled by authors as a generic training process that involves minimization of a rank-based loss function, similarly to TransE [22] and other approaches.

KALE A novel embedding knowledge graphs strategy – namely KALE – is proposed [79]. The proposed framework does not rely solely on knowledge graphs like TransE [22], but it considers using additional information available under the form of logic rules. Indeed, such additional information can be beneficial for the embedding learning task. Knowledge graphs are embedded into continuous vector spaces through minimization of a custom margin-based ranking loss function. The proposed loss function accounts for both atomic and complex formulae. Atomic formulae represent fact triplets already available in the knowledge graph, while complex formulae represents additional logical rules at hand. The margin-based loss function enforces positive formulae to have larger truth values than negative ones. The proposed embedding seems compatible not only with triplets but also with logic rules. KALE leverages t-norm fuzzy logic techniques to model logic rules and produce formulae based on them.

HolE Authors propose a novel method for embedding knowledge graphs [131]. The proposed method relies on Holographic Embeddings – thus HolE – to learn compositional vector space representations of entire knowledge graphs. Holographic embedding relies on circular correlation to create compositional representations of relations between entities. Entity embeddings are composed using circular convolutions and Hadamard product to obtain the circular correlation representing relations between entities. Both entity embeddings and circular correlation representing relations are then optimised during training via minimization of either a (regularized) logistic loss or a pairwise ranking loss.

ComplEx Authors propose a novel knowledge graph embedding mechanism, which relies on factorization of entities and relations through complex valued vectors—thus ComplEx [187]. Complex valued vectors are used since their composition can represent a large variety of binary relations, like symmetry and antisymmetry. Similarly to previous knowledge graph embedding mechanisms – TransE [22], TransH [199], etc. –, the embedding operation is trained via minimization of a custom scoring function. Authors propose to define such custom scoring function as the real component of the inner product between the first entity embedding, the relation embeddings, and the embedding of the second entity complex conjugate.

GCN Graph Convolutional Networks (GCN) are introduced in [98]. A novel convolutional paradigm over graph structured data is proposed to embed, manage and manipulate knowledge available over graphs. Convolution operation is applied to graph leveraging a first-order approximation of spectral convolutions which rely on the use of message passing and aggregation of neighborhood information. The graph learning task is mapped to a semi-supervised node classification task, where a graph-based regularization loss term is leveraged during training to enhance graph structured learning. Such techniques is capable of correctly learning graph-structured knowledge, showing encouraging results over nodes classification tasks.

Analogy Authors in [111] present a novel knowledge graph embedding framework which aims at explicitly modelling analogical structures—thus the name. Analogies between entities available in a knowledge base are mapped into normal transformations—i.e., transformations obtained via the application of normal matrices. A differentiable objective function and a linear-time inference algorithm are also proposed to learn such normal transformations. Moreover, authors show that the proposed framework is a generalization of state-of-the-art embedding mechanisms that rely on linear transformations, like ComplEx [187], HolE [131], DistMult [209], etc.

GSNN Authors introduce Graph Search Neural Networks (GSNN), a novel technique to incorporate prior knowledge into NNs used for image classification [115]. The prior knowledge is considered to be available under the form of knowledge graphs. The graph search neural network, given an input image, detects the concepts available in the image via means of object detection. Such concepts are then compared with the knowledge graph at hand in order to expand detected entities with neighbouring entities. The information of neighbouring entities is then used to obtain an embedding of the relevant prior knowledge via means of Graph Neural Networks. Such embedding is then used as input to a classification layer to obtain the final prediction for the considered image.

NPE Authors introduce the Neural Physics Engine (NPE), a differentiable physics simulator that combines rough symbolic structure with gradient-based learning [35]. Notions of objects-specific properties and object interactions are explicitly represented as symbolic knowledge. This knowledge and notions are then embedded – via vectorization – into a Neural Network consisting of the main core of the NPE. Physical notions are represented as state vectors of the objects involved in the simulation. Therefore, the vectorization process is rather straightforward. The NN is then tailored to suit specific object properties and dynamics of a given world through training.

GRAM Authors in [41] propose a graph-based attention model for prediction of diseases—namely GRAM. The GRAM framework injects domain clinical knowledge which is available under the form of clinical ontologises, represented as directed acylcic graphs. In these ontologies, clinical concepts are releated via parent-children relations, where parent clinical concepts are more general, while children are more specific. An attention mechanism is then used to obtain the vectorial representation of a clinical concept from the ontology graph. More in details, the concept is combined with its ancestors, via attention and weighted sum, to obtain its final vectorial representation. Finally, vectorial representation of clinical concepts are concatenated – vectorially – and passed to the predictor NN, to infer possible diseases from symptoms.

GAT Authors present Graph Attention Networks (GAT) as a novel technique to embed and treat graph-structured data sub-symbolically [194]. More in details, an attention mechanism is provided to weight information of neighbouring nodes in the graph convolution mechanism (GCN) [98]. This attention mechanism provides significant improvements over standard graph convolution. Authors leverage the proposed framework to classify each node in the graph structure at hand. However, this approach can be generally useful to treat knowledge available under the form of knowledge graphs.

CCM Authors present the common sense knowledge aware Conversational Model (CCM) [217]. The proposed work aims at demonstrating how common-sense knowledge can facilitate generation of text replies in conversational systems. Given a text message, the proposed framework samples the entities belonging to the sentence and identify knowledge graphs corresponding to the entities at hand. These knowledge graphs are then embedded under vectorial form through static/dynamic graph attention mechanism [194]. The obtained common sense knowledge vector is then used as the input vector for a NN that generates the reply for the given input sentence.

RotatE Authors propose RotatE, a novel relations embedding model, which focus on complex relations characteristics such as symmetry, inversion and composition [172]. Such properties are not fully considered by previous approaches such as TransE [22], TransH [199], etc. Therefore, authors propose to model relationships as rotations from the source entity to the target entity over the embedding space. Indeed, rotation is very well suited for mapping some properties like symmetry, inversion and composition. Authors express the rotation operation as a Hadamard – or element-wise – product between an entity embedding and a relation embedding, where the latter has a fixed unitary modulus. Similarly to TransE and TransH, the embedding operation is trained via minimization of a custom loss function. Finally, authors propose a novel adversarial sampling of negative triplets examples, which is leveraged to optimise the training procedure.

HAKE Zhang et al. [215] propose a Hierarchy-Aware Knowledge graph Embedding (HAKE), a novel knowledge graph embedding model, which aims at modelling semantic hierarchies. Such hierarchies are common in real-world applications and are not considered by previous approaches like TransE [22]. The building concept of this model is to map entities and relations to polar coordinates. In the polar coordinates space, concentric circles reflect the knowledge of hierarchies. More specifically, hierarchy levels are modelled via radial coordinates, entities with smaller radii are expected to be at higher levels of hierarchy, being closer to the origin of the polar coordinates space. On the other hand, angular coordinates model entities at the same level of the hierarchy, and are used to distinguish between them. Finally, similarly to previous knowledge graph embedding mechanisms, the embedding operation is trained via minimization of a custom loss function.

References

[1] Andrea Agiollo, Giovanni Ciatto, and Andrea Omicini. Graph neural networks as the copula mundi between logic and machine learning: A roadmap. In Roberta Calegari, Giovanni Ciatto, Enrico Denti, Andrea Omicini, and Giovanni Sartor, editors, *WOA 2021 – 22nd Workshop "From Objects to Agents"*, volume 2963 of *CEUR Workshop Proceedings*, pages 98–115. Sun SITE Central Europe, RWTH Aachen University, October 2021. 22nd Workshop "From Objects to Agents" (WOA 2021), Bologna, Italy, 1–3 September 2021. Proceedings.

[2] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, NSW, Australia, August 6-11 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 80–88. PMLR, 2017.

[3] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[4] Robert Andrews and Shlomo Geva. Rulex & cebp networks as the basis for a rule refinement system. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 1–12. IOS Press, 1995.

[5] M. Gethsiyal Augasta and T. Kathirvalavakumar. Reverse engineering the neural networks for rule extraction in classification problems. *Neural Process. Lett.*, 35(2):131–150, 2012.

[6] Franz Baader. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95, USA, 2003. Cambridge University Press.

[7] Sebastian Bader. Extracting propositional rules from feedforward neural networks by means of binary decision diagrams. In Artur S. d'Avila Garcez and Pascal Hitzler, editors, *Proceedings of the Fifth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy 2009, Pasadena, CA, USA, July 11, 2009*, volume 481 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[8] Sebastian Bader, Artur S. d'Avila Garcez, and Pascal Hitzler. Computing first-order logic programs by fibring artificial neural networks. In Ingrid Russell and Zdravko Markov, editors, *Proceedings of the 18th International Florida Artificial*

Intelligence Research Society Conference (FlAIRS), Clearwater Beach, Florida, USA, May 15–17, 2005, pages 314–319. AAAI Press, 2005.

[9] Sebastian Bader, Steffen Hölldobler, and Nuno C. Marques. Guiding backprop by inserting rules. In Artur S. d'Avila Garcez and Pascal Hitzler, editors, *Proceedings of the 4th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy), Patras, Greece, July 21, 2008*, volume 366 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[10] Sebastian Bader, Steffen Hölldobler, and Valentin Mayer-Eichberger. Extracting propositional rules from feed-forward neural networks – A new decompositional approach. In Artur S. d'Avila Garcez, Pascal Hitzler, and Guglielmo Tamburrini, editors, *Proceedings of the 3rd International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'07, held at IJCAI-07, Hyderabad, India, January 8, 2007*, volume 230 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[11] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *CoRR*, abs/2012.13635, 2020.

[12] Dana H. Ballard. Parallel logical inference and energy minimization. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 203–209. Morgan Kaufmann, 1986.

[13] Nahla H. Barakat and Andrew P. Bradley. Rule extraction from support vector machines: A sequential covering approach. *IEEE Trans. Knowl. Data Eng.*, 19(6):729–741, 2007.

[14] Nahla H. Barakat and Joachim Diederich. Eclectic rule-extraction from support vector machines. *International Journal of Computer and Information Engineering*, 2(5):1672–1675, 2008.

[15] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58(December 2019):82–115, 2020.

[16] Rodrigo Basilio, Gerson Zaverucha, and Valmir Carneiro Barbosa. Learning logic programs with neural networks. In Céline Rouveirol and Michèle Sebag, editors, *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings*, volume 2157 of *Lecture Notes in Computer Science*, pages 15–26. Springer, 2001.

[17] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. Are artificial neural networks black boxes? *IEEE Trans. Neural Networks*, 8(5):1156–1164, 1997.

[18] Hamid R. Berenji. Refinement of approximate reasoning-based controllers by reinforcement learning. In Lawrence Birnbaum and Gregg Collins, editors, *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pages 475–479. Morgan Kaufmann, 1991.

[19] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902, 2017.

[20] Jiang Bian, Bin Gao, and Tie-Yan Liu. Knowledge-powered deep learning for word embedding. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Proceedings of the 25th Machine Learning and Knowledge Discovery in Databases - European Conference (ECML), Nancy, France, September 15-19, 2014*, volume 8724 of *Lecture Notes in Computer Science*, pages 132–148. Springer, 2014.

[21] S. Blackburn. *The Oxford Dictionary of Philosophy.* Oxford Paperback Reference. OUP Oxford, 2008.

[22] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Proceedings of 27th Annual Conference on Neural Information Processing Systems (NeurIPS), Lake Tahoe, Nevada, United States, December 5-8, 2013*, pages 2787–2795, 2013.

[23] Olcay Boz. Converting a trained neural network to a decision tree DecText - decision tree extractor. In M. Arif Wani, Hamid R. Arabnia, Krzysztof J. Cios, Khalid Hafeez, and Graham Kendall, editors, *Proceedings of the 2002 International Conference on Machine Learning and Applications - ICMLA 2002, June 24-27, 2002, Las Vegas, Nevada, USA*, pages 110–116. CSREA Press, 2002.

[24] Ronald J. Brachman and Hector J. Levesque. The tradeoff between expressiveness and tractability. In Ronald J. Brachman and Hector J. Levesque, editors, *Knowledge Representation and Reasoning*, The Morgan Kaufmann Series in Artificial Intelligence, pages 327–348. Morgan Kaufmann, San Francisco, 2004.

[25] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees.* CRC Press, 1984.

[26] Antony Browne, Brian D. Hudson, David C. Whitley, Martyn G. Ford, and Philip Picton. Biological data mining with neural networks: implementation and application of a flexible decision tree extraction algorithm to genomic problem domains. *Neurocomputing*, 57:275–293, March 2004.

[27] Clifford Brunk and Michael J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In Lawrence Birnbaum and Gregg Collins, editors, *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*, pages 389–393. Morgan Kaufmann, 1991.

[28] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32, 2020.

[29] Davide Calvaresi, Giovanni Ciatto, Amro Najjar, Reyhan Aydoğan, Leon Van der Torre, Andrea Omicini, and Michael Schumacher. EXPECTATION: Personalized explainable artificial intelligence for decentralized agents with heterogeneous knowledge. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *LNCS*, pages 331–343. Springer Nature, Basel, Switzerland, 2021.

[30] Gail A. Carpenter, Stephen Grossberg, Natalya Markuzon, John H. Reynolds, and David B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transaction on Neural Networks*, 3(5):698–713, 1992.

[31] Luis A. Castillo, Antonio González Muñoz, and Raúl Pérez. Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm. *Fuzzy Sets Syst.*, 120(2):309–321, 2001.

[32] Veronica Chan and Christine W. Chan. Towards developing the piece-wise linear neural network algorithm for rule extraction. *Int. J. Cogn. Informatics Nat. Intell.*, 11(2):57–73, 2017.

[33] Veronica K.H. Chan and Christine W. Chan. Towards explicit representation of an artificial neural network model: Comparison of two artificial neural network rule extraction approaches. *Petroleum*, 6(4):329–339, 2020. SI: Artificial Intelligence (AI), Knowledge-based Systems (KBS), and Machine Learning (ML).

[34] Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, October 25-29, 2014*, pages 1568–1579. ACL, 2014.

[35] Michael Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017*. OpenReview.net, 2017.

[36] Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. Guiding semi-supervision with constraint-driven learning. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL), June 23-30, 2007, Prague, Czech Republic.* The Association for Computational Linguistics (ACL), 2007.

[37] Adriana da Costa F Chaves, Marley M. B. R. Vellasco, and Ricardo Tanscheit. Fuzzy rule extraction from support vector machines. In Nadia Nedjah, Luiza de Macedo Mourelle, Ajith Abraham, and Mario Köppen, editors, *5th International Conference on Hybrid Intelligent Systems (HIS 2005), 6-9 November 2005, Rio de Janeiro, Brazil*, pages 335–340. IEEE Computer Society, 2005.

[38] Zhengping Che, David C. Kale, Wenzhe Li, Mohammad Taha Bahadori, and Yan Liu. Deep computational phenotyping. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (KDD), Sydney, NSW, Australia, August 10-13, 2015*, pages 507–516. ACM, 2015.

[39] Fei Chen. Learning accurate and understandable rules from svm classifiers. Master's thesis, 2004.

[40] Zhenyu Chen, Jianping Li, and Liwei Wei. A multiple kernel support vector machine scheme for feature selection and rule extraction from gene expression data of cancer tissue. *Artif. Intell. Medicine*, 41(2):161–175, 2007.

[41] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F. Stewart, and Jimeng Sun. GRAM: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Halifax, NS, Canada, August 13-17, 2017*, pages 787–795. ACM, 2017.

[42] Shivani Choudhary, Tarun Luthra, Ashima Mittal, and Rajat Singh. A survey of knowledge graph embedding and their applications. *CoRR*, abs/2107.07842, 2021.

[43] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[44] Giovanni Ciatto, Roberta Calegari, Andrea Omicini, and Davide Calvaresi. Towards XMAS: eXplainability through Multi-Agent Systems. In Claudio Savaglio, Giancarlo Fortino, Giovanni Ciatto, and Andrea Omicini, editors, *AI&IoT 2019 – Artificial Intelligence and Internet of Things 2019*, volume 2502 of *CEUR Workshop Proceedings*, pages 40–53. CEUR WS, November 2019.

[45] Giovanni Ciatto, Davide Calvaresi, Michael I. Schumacher, and Andrea Omicini. An abstract framework for agent-based explanations in AI. In Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith, editors, *19th*

International Conference on Autonomous Agents and MultiAgent Systems, pages 1816–1818. IFAAMAS, May 2020.

[46] Giovanni Ciatto, Michael I. Schumacher, Andrea Omicini, and Davide Calvaresi. Agent-based explanations in AI: Towards an abstract framework. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, volume 12175 of *LNCS*, pages 3–20. Springer, Cham, 2020.

[47] Philipp Cimiano. *Ontology Learning and Population from Text.* Springer US, 2006.

[48] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Mach. Learn.*, 3:261–283, 1989.

[49] William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 988–994. Morgan Kaufmann, 1993.

[50] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart J. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 115–123. Morgan Kaufmann, 1995.

[51] James W. Cooley, Peter A. W. Lewis, and Peter D. Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.

[52] Mark W. Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning Proceedings 1994*, pages 37–45. Elsevier, 1994.

[53] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, pages 24–30. The MIT Press, June 1996.

[54] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artif. Intell.*, 125(1-2):155–207, 2001.

[55] Artur S. d'Avila Garcez and Dov M. Gabbay. Fibring neural networks. In Deborah L. McGuinness and George Ferguson, editors, *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*, pages 342–347. AAAI Press / The MIT Press, 2004.

[56] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Appl. Intell.*, 11(1):59–77, 1999.

[57] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 1389–1399. The Association for Computational Linguistics, 2016.

[58] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota, USA, jun 2019. Association for Computational Linguistics (ACL).

[59] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

[60] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In Xuewen Chen, Bo Luo, Feng Luo, Vasile Palade, and M. Arif Wani, editors, *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, December 18-21, 2017*, pages 920–923. IEEE, 2017.

[61] Terence A. Etchells and Lisboa Paulo J. G. Orthogonal search-based rule extraction (OSRE) for trained neural networks: a practical and efficient approach. *IEEE Trans. Neural Networks*, 17(2):374–384, 2006.

[62] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Jounal of Artificial Intelligence Research*, 61:1–64, 2018.

[63] Yuan Fang, Kingsley Kuan, Jie Lin, Cheston Tan, and Vijay Chandrasekhar. Object detection meets knowledge graphs. In Carles Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), Melbourne, Australia, August 19-25, 2017*, pages 1661–1667. IJCAI, 2017.

[64] Manoel V. M. França, Gerson Zaverucha, and Artur S. d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94(1):81–104, 2014.

[65] Ray J. Frank, Neil Davey, and Stephen P. Hunt. Time series prediction and neural networks. *J. Intell. Robotic Syst.*, 31(1-3):91–103, 2001.

[66] Alex A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, June 2014.

[67] LiMin Fu. Rule generation from neural networks. *IEEE Trans. Syst. Man Cybern. Syst.*, 24(8):1114–1124, 1994.

[68] Xiuju Fu, ChongJin Ong, S. Keerthi, Gih Guang Hung, and Liping Goh. Extracting the knowledge embedded in support vector machines. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 1, pages 291–296, 2004.

[69] Glenn Fung, Sathyakama Sandilya, and R. Bharat Rao. Rule extraction from linear support vector machines. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 32–40. ACM, 2005.

[70] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 70–77. Morgan Kaufmann, 1994.

[71] Alberto García-Durán, Antoine Bordes, and Nicolas Usunier. Effective blending of two and three-way interactions for modeling multi-relational data. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Proceeding of the 25th Machine Learning and Knowledge Discovery in Databases - European Conference (ECML), Nancy, France, September 15-19, 2014*, volume 8724 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2014.

[72] C. Lee Giles and Christian W. Omlin. Rule refinement with recurrent neural networks. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 801–806. IEEE, 1993.

[73] Goran Glavas and Ivan Vulic. Explicit retrofitting of distributional word vectors. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Melbourne, Australia, July 15-20, 2018*, pages 34–45. Association for Computational Linguistics, 2018.

[74] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[75] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

[76] Travis R. Goodwin and Dina Demner-Fushman. Bridging the knowledge gap: Enhancing question answering with world and domain knowledge. *CoRR*, abs/1910.07429, 2019.

[77] C. Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In *1968 23rd ACM National Conference*, pages 169–181, 1968.

[78] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, 2018.

[79] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 192–202. The Association for Computational Linguistics, 2016.

[80] Tameru Hailesilassie. Rule extraction algorithm for deep neural networks: A review. *CoRR*, abs/1610.05267, 2016.

[81] Saman K. Halgamuge and Manfred Glesner. Neural networks in designing fuzzy systems for real world applications. *Fuzzy Sets and Systems*, 65(1):1–12, 1994.

[82] Yoichi Hayashi. A neural expert system with automated extraction of fuzzy if-then rules. In Richard Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems 3, [NIPS Conference, Denver, Colorado, USA, November 26-29, 1990]*, pages 578–584. Morgan Kaufmann, 1990.

[83] Jieyue He, Hae-Jin Hu, R. Harrison, P.C. Tai, and Yi Pan. Rule generation for protein secondary structure prediction with support vector machines and decision tree. *IEEE Transactions on NanoBioscience*, 5(1):46–53, 2006.

[84] Tzung-Pei Hong and Jyh-Bin Chen. Finding relevant attributes and membership functions. *Fuzzy Sets Syst.*, 103(3):389–404, 1999.

[85] Tzung-Pei Hong and Jyh-Bin Chen. Processing individual fuzzy attributes for fuzzy rule induction. *Fuzzy Sets Syst.*, 112(1):127–140, 2000.

[86] Tzung-Pei Hong and Chai-Ying Lee. Induction of fuzzy rules and membership functions from training examples. *Fuzzy Sets Syst.*, 84(1):33–47, 1996.

[87] Shin-ichi Horikawa, Takeshi Furuhashi, and Yoshiki Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Trans. Neural Networks*, 3(5):801–806, 1992.

[88] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951.

[89] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. 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.

[90] Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric P. Xing. Deep neural networks with massive learned knowledge. In Jian Su, Xavier Carreras, and

Kevin Duh, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016*, pages 1670–1679. The Association for Computational Linguistics, 2016.

[91] Johan Huysmans, Bart Baesens, and Jan Vanthienen. ITER: An algorithm for predictive regression rule extraction. In *Data Warehousing and Knowledge Discovery (DaWaK 2006)*, pages 270–279. Springer, 2006.

[92] Johan Huysmans, Bart Baesens, and Jan Vanthienen. Using rule extraction to improve the comprehensibility of predictive models. *Behavioral & Experimental Economics*, 2006.

[93] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[94] Hisao Ishibuchi, Manabu Nii, and Tadahiko Murata. Linguistic rule extraction from neural networks and genetic-algorithm-based rule selection. In *Proceedings of International Conference on Neural Networks (ICNN'97), Houston, TX, USA, June 9-12, 1997*, pages 2390–2395. IEEE, 1997.

[95] DaeEun Kim and Jaeho Lee. Handling continuous-valued attributes in decision tree with neural network modeling. In Ramon López de Mántaras and Enric Plaza, editors, *Machine Learning: ECML 2000*, pages 211–219, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[96] Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. pages 1–4. Mansinghka, Vikash, 2012.

[97] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. An algebraic prolog for reasoning about possible worlds. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the 25th Conference on Artificial Intelligence (AAAI), San Francisco, California, USA, August 7-11, 2011.* AAAI Press, 2011.

[98] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017.* OpenReview.net, 2017.

[99] Rikard Konig, Ulf Johansson, and Lars Niklasson. G-REX: A versatile framework for evolutionary data mining. In *2008 IEEE International Conference on Data Mining Workshops (ICDM 2008 Workshops)*, pages 971–974, 2008.

[100] Sotiris Kotsiantis. Supervised machine learning: A review of classification techniques. In *Emerging Artificial Intelligence Applications in Computer Engineering*, volume 160 of *Frontiers in Artificial Intelligence and Applications*, pages 3–24. IOS Press, October 2007.

[101] R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognit.*, 32(12):1999–2009, 1999.

[102] R. Krishnan, G. Sivakumar, and P. Bhattacharya. A search technique for rule extraction from trained neural networks. *Pattern Recognit. Lett.*, 20(3):273–280, 1999.

[103] Luís C. Lamb, Artur S. d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4877–4884. ijcai.org, 2020.

[104] Jens Lehmann, Sebastian Bader, and Pascal Hitzler. Extracting reduced logic programs from artificial neural networks. *Appl. Intell.*, 32(3):249–266, 2010.

[105] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Comput. Intell.*, 3:78–93, 1987.

[106] Xiaodan Liang, Zhiting Hu, Hao Zhang, Liang Lin, and Eric P. Xing. Symbolic graph reasoning meets convolutions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS), Montréal, Canada, December 3-8, 2018*, pages 1858–1868, 2018.

[107] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the 29th Conference on Artificial Intelligence (AAAI), Austin, Texas, USA, January 25-30, 2015*, pages 2181–2187. AAAI Press, 2015.

[108] Zachary C. Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, June 2018.

[109] Bo Liu, Hussein A. Abbass, and Robert I. McKay. Density-based heuristic for rule discovery with ant-miner. In *The 6th Australia-Japan joint workshop on intelligent and evolutionary system*, volume 184, 2002.

[110] Bo Liu, Hussein A. Abbass, and Robert I. McKay. Classification rule discovery with ant colony optimization. *IEEE Intell. Informatics Bull.*, 3(1):31–35, 2004.

[111] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, NSW, Australia, August 6-11, 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2168–2178. PMLR, 2017.

[112] Tianle Ma and Aidong Zhang. Multi-view factorization autoencoder with network constraints for multi-omic integrative analysis. In Huiru Jane Zheng, Zoraida Callejas, David Griol, Haiying Wang, Xiaohua Hu, Harald H. H. W. Schmidt, Jan Baumbach, Julie Dickerson, and Le Zhang, editors, *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Madrid, Spain, December 3-6, 2018*, pages 702–707. IEEE Computer Society, 2018.

[113] J. A. Makowsky. Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34(2):266–292, 1987.

[114] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021.

[115] Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. The more you know: Using knowledge graphs for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 21-26, 2017*, pages 20–28. IEEE Computer Society, 2017.

[116] Urszula Markowska-Kaczmar and Marcin Chumieja. Discovering the mysteries of neural networks. *Int. J. Hybrid Intell. Syst.*, 1(3-4):153–163, 2004.

[117] Urszula Markowska-Kaczmar and Wojciech Trelak. Extraction of fuzzy rules from trained neural network using evolutionary algorithm. In *ESANN 2003, 11th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 23-25, 2003, Proceedings*, pages 149–154, 2003.

[118] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: a general interface layer to integrate AI and deep learning. *CoRR*, abs/1903.07534, 2019.

[119] David Martens, Bart Baesens, and Tony Van Gestel. Decompositional rule extraction from support vector machines by active learning. *IEEE Trans. Knowl. Data Eng.*, 21(2):178–191, 2009.

[120] David Martens, Manu De Backer, Raf Haesen, Jan Vanthienen, Monique Snoeck, and Bart Baesens. Classification with ant colony optimization. *IEEE Trans. Evol. Comput.*, 11(5):651–665, 2007.

[121] R. Masuoka, N. Watanabe, A. Kawamura, Y. Owada, and K. Asakawa. Neurofuzzy systems – Fuzzy inference using a structured neural network. In *Proceedings of International Conference on Fuzzy Logic and Neural Networks, Iizuka Japan, July, 1990*, pages 173–177, 1990.

[122] Chirs Matthews and Ilona Jagielska. Fuzzy rule extraction from a trained multilayer neural network. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 2, pages 744–748 vol.2, 1995.

[123] George F. McNulty. Fragments of first order logic, i: Universal Horn logic. *Journal of Symbolic Logic*, 42(2):221–237, 1977.

[124] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In Keh-Yih Su, Jian Su, and Janyce Wiebe, editors, *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Singapore, August 2-7, 2009*, pages 1003–1011. The Association for Computer Linguistics (ACL), 2009.

[125] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[126] Sushmita Mitra. Fuzzy mlp based expert system for medical diagnosis. *Fuzzy Sets and Systems*, 65(2):285–296, 1994. Fuzzy Methods for Computer Vision and Pattern Recognition.

[127] Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Counter-fitting word vectors to linguistic constraints. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), San Diego California, USA, June 12-17, 2016*, pages 142–148. The Association for Computational Linguistics, 2016.

[128] Patrick M. Murphy and Michael J. Pazzani. Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Machine Learning Proceedings 1991*, pages 183–187. Elsevier, 1991.

[129] Detlef D. Nauck and Rudolf Kruse. A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets Syst.*, 89(3):277–288, 1997.

[130] Detlef D. Nauck and Rudolf Kruse. Neuro-fuzzy systems for function approximation. *Fuzzy Sets Syst.*, 101(2):261–271, 1999.

[131] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the 30th Conference on Artificial Intelligence (AAAI), Phoenix, Arizona, USA, February 12-17, 2016*, pages 1955–1961. AAAI Press, 2016.

[132] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML), Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress, 2011.

[133] Haydemar Núñez, Cecilio Angulo, and Andreu Català. Rule extraction based on support and prototype vectors. In Joachim Diederich, editor, *Rule Extraction from Support Vector Machines*, volume 80 of *Studies in Computational Intelligence*, pages 109–134. Springer, 2008.

[134] Koichi Odajima, Yoichi Hayashi, Gong Tianxia, and Rudy Setiono. Greedy rule generation from discrete data and its use in neural network rule extraction. *Neural Networks*, 21(7):1020–1028, 2008.

[135] Rafael S. Parpinelli, Heitor S. Lopes, and Alex A. Freitas. An ant colony based system for data mining: applications to medical data. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, pages 791–797. Citeseer, 2001.

[136] Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, November 3-7, 2019*, pages 43–54. Association for Computational Linguistics, 2019.

[137] Gadi Pinkas, Priscila Lima, and Shimon Cohen. Representing, binding, retrieving and unifying relational knowledge using pools of neural binders. *Biologically Inspired Cognitive Architectures*, 6:87–95, 2013. BICA 2013: Papers from the Fourth Annual Meeting of the BICA Society.

[138] Gadi Pinkus. Constructing proofs in symmetric networks. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 217–224. Morgan Kaufmann, 1991.

[139] Emanoil Pop, Ross Hayward, and Joachim Diederich. RULENEG: Extracting rules from a trained ANN by stepwise negation. Technical report, Neurocomputing Research Centre, Queensland University of Technology, 1994.

[140] J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

[141] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[142] J. Ross Quinlan. C4.5: Programming for machine learning. *Morgan Kauffmann*, 1993.

[143] Juan R. Rabuñal, Julian Dorado, Alejandro Pazos, Javier Pereira, and Daniel Rivero. A new approach to the extraction of ANN rules and to their generalization capacity through GP. *Neural Comput.*, 16(7):1483–1523, 2004.

[144] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA, December 4-9, 2017*, pages 3788–3800, 2017.

[145] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *The 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies (HLT), Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129. The Association for Computational Linguistics, 2015.

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

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

[148] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[149] Emad W. Saad and Donald C. Wunsch II. Neural network explanation using inversion. *Neural Networks*, 20(1):78–93, 2007.

[150] Federico Sabbatini, Giovanni Ciatto, and Andrea Omicini. GridEx: An algorithm for knowledge extraction from black-box regressors. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *LNCS*, pages 18–38. Springer Nature, Basel, Switzerland, 2021.

[151] Kazumi Saito and Ryohei Nakano. Medical diagnostic expert system based on PDP model. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Diego, CA, USA, July 24-27, 1988*, pages 255–262. IEEE, 1988.

[152] Kazumi Saito and Ryohei Nakano. Law discovery using neural networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 1078–1083. Morgan Kaufmann, 1997.

[153] Kazumi Saito and Ryohei Nakano. Extracting regression rules from neural networks. *Neural Networks*, 15(10):1279–1288, 2002.

[154] Makoto Sato and Hiroshi Tsukimoto. Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1870–1875. IEEE, 2001.

[155] Vitaly Schetinin, Jonathan E. Fieldsend, Derek Partridge, Timothy J. Coats, Wojtek J. Krzanowski, Richard M. Everson, Trevor C. Bailey, and Adolfo Hernandez. Confident interpretation of bayesian decision tree ensembles for clinical applications. *IEEE Trans. Inf. Technol. Biomed.*, 11(3):312–319, 2007.

[156] Gregor P. J. Schmitz, Chris Aldrich, and François S. Gouws. ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6):1392–1401, 1999.

[157] Sabrina Sestito and Tharam S. Dillon. *Automated knowledge acquisition.* Prentice Hall International series in computer science and engineering. Prentice Hall, 1994.

[158] Kamal Kumar Sethi, Durgesh Kumar Mishra, and Bharat Mishra. KDRuleEx: A novel approach for enhancing user comprehensibility using rule extraction. In *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, pages 55–60, 2012.

[159] Rudy Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Comput.*, 9(1):205–225, 1997.

[160] Rudy Setiono. Extracting M-of-N rules from trained neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 11(2):512–519, 2000.

[161] Rudy Setiono, Bart Baesens, and Christophe Mues. Recursive neural network rule extraction for data with mixed attributes. *IEEE Trans. Neural Networks*, 19(2):299–307, 2008.

[162] Rudy Setiono and Wee Kheng Leow. FERNN: An algorithm for fast extraction of rules from neural networks. *Appl. Intell.*, 12(1-2):15–25, 2000.

[163] Rudy Setiono, Wee Kheng Leow, and Jacek M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, 13(3):564–577, 2002.

[164] Rudy Setiono and Huan Liu. Symbolic representation of neural networks. *Computer*, 29(3):71–77, 1996.

[165] Rudy Setiono and Huan Liu. Neurolinear: A system for extracting oblique decision rules from neural networks. In Maarten van Someren and Gerhard Widmer, editors, *Machine Learning: ECML-97, 9th European Conference on Machine Learning, Prague, Czech Republic, April 23-25, 1997, Proceedings*, volume 1224 of *Lecture Notes in Computer Science*, pages 221–233. Springer, 1997.

[166] Rudy Setiono and Huan Liu. NeuroLinear: From neural networks to oblique decision rules. *Neurocomputing*, 17(1):1–24, 1997.

[167] Rudy Setiono and James Y. L. Thong. An approach to generate rules from neural networks for regression problems. *Eur. J. Oper. Res.*, 155(1):239–250, 2004.

[168] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46(1-2):159–216, 1990.

[169] Raymond M. Smullyan. *First-Order Logic.* New York [Etc.]Springer-Verlag, 1968.

[170] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezný, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[171] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the 31st Conference on Artificial Intelligence (AAAI), San Francisco, California, USA, February 4-9, 2017*, pages 2576–2582. AAAI Press, 2017.

[172] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net, 2019.

[173] Ismail A. Taha and Joydeep Ghosh. Symbolic interpretation of artificial neural networks. *IEEE Trans. Knowl. Data Eng.*, 11(3):448–463, 1999.

[174] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Beijing, China, July 26-31, 2015*, pages 1556–1566. The Association for Computer Linguistics, 2015.

[175] Ah-Hwee Tan. Cascade ARTMAP: integrating neural computation and symbolic knowledge processing. *IEEE Transaction on Neural Networks*, 8(2):237–250, 1997.

[176] Sebastian B. Thrun. Extracting provably correct rules from artificial neural networks. Technical report, University of Bonn, 1993.

[177] Sebastian B. Thrun. Extracting rules from artifical neural networks with distributed representations. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 505–512. MIT Press, 1994.

[178] Alan B. Tickle, Marian Orlowski, and Joachim Diederich. DEDEC: A methodology for extracting rules from trained artificial neural networks. In Robert Andrews and Joachim Diederich, editors, *Rules and Networks: Proceedings of the Rule Extraction from Trained Artificial Neural Networks Workshop*, pages 90–102. Neurocomputing Research Centre, Queensland University of Technology, 1996.

[179] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 465–474. ACM, 2017.

[180] Douglas E. D. Torres and Claudio M. S. Rocco. Extracting trees from trained SVM models using a TREPAN based approach. In Nadia Nedjah, Luiza de Macedo Mourelle, Ajith Abraham, and Mario Köppen, editors, *5th International Conference on Hybrid Intelligent Systems (HIS 2005), 6-9 November 2005, Rio de Janeiro, Brazil*, pages 353–360. IEEE Computer Society, 2005.

[181] Geoffrey G. Towell and Jude W. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 977–984. Morgan Kaufmann, 1991.

[182] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[183] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement ofapproximate domain theories by knowledge-based neural networks. In Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*, pages 861–866. AAAI Press / The MIT Press, 1990.

[184] Son N. Tran and Artur S. d'Avila Garcez. Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE Transaction on Neural Networks and Learning Systems*, 29(2):246–258, 2016.

[185] Volker Tresp, Jürgen Hollatz, and Subutai Ahmad. Network structuring and training using rule-based knowledge. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 871–878. Morgan Kaufmann, 1992.

[186] Volker Tresp, Jürgen Hollatz, and Subutai Ahmad. Network structuring and training using rule-based knowledge. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5,*

[NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992], pages 871–878. Morgan Kaufmann, 1992.

[187] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML), New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016.

[188] Hiroshi Tsukimoto. Extracting rules from trained neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 11(2):377–389, 2000.

[189] Bhekisipho Twala. Multiple classifier application to credit risk assessment. *Expert Systems with Applications*, 37(4):3326–3336, 2010.

[190] Johan Van Benthem and Kees Doets. *Higher-Order Logic*, pages 189–243. Springer Netherlands, Dordrecht, 2001.

[191] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, October 1976.

[192] Tim van Gelder. Why distributed representation is inherently non-symbolic. In Georg Dorffner, editor, *Konnektionismus in Artificial Intelligence und Kognitionsforschung. Proceedings 6. Österreichische Artificial Intelligence-Tagung (KONNAI), Salzburg, Österreich, 18. bis 21. September 1990*, volume 252 of *Informatik-Fachberichte*, pages 58–66. Springer, 1990.

[193] F. Van Veen and S. Leijnen. The neural network zoo. `https://www.asimovinstitute.org/neural-network-zoo`, 2019. [Online; accessed 17-September-2021].

[194] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, April 30 - May 3, 2018.* OpenReview.net, 2018.

[195] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR). A Practical Guide.* Springer, 2017.

[196] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[197] Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina, July 25-31, 2015*, pages 1859–1866. AAAI Press, 2015.

[198] Sutong Wang, Yuyan Wang, Dujuan Wang, Yunqiang Yin, Yanzhang Wang, and Yaochu Jin. An improved random forest-based rule extraction method for breast cancer diagnosis. *Appl. Soft Comput.*, 86, 2020.

[199] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th Conference on Artificial Intelligence (AAAI), Québec City, Québec, Canada, July 27 -31, 2014*, pages 1112–1119. AAAI Press, 2014.

[200] Zhuoyu Wei, Jun Zhao, Kang Liu, Zhenyu Qi, Zhengya Sun, and Guanhua Tian. Large-scale knowledge base completion: Inferring via grounding network sampling over selected instances. In James Bailey, Alistair Moffat, Charu C. Aggarwal, Maarten de Rijke, Ravi Kumar, Vanessa Murdock, Timos K. Sellis, and Jeffrey Xu Yu, editors, *Proceedings of the 24th International Conference on Information and Knowledge Management (CIKM), Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 1331–1340. ACM, 2015.

[201] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015*, 2015.

[202] Wikipedia contributors. Activation function — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Activation_function`, 2021. [Online; accessed 23-August-2021].

[203] Wikipedia contributors. Decision tree learning — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Decision_tree_learning`, 2021. [Online; accessed 17-September-2021].

[204] Wikipedia contributors. Stochastic gradient descent — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent`, 2021. [Online; accessed 23-August-2021].

[205] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82, 1997.

[206] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks Learning Systems*, 32(1):4–24, 2021.

[207] Yaqi Xie, Ziwei Xu, Kuldeep S. Meel, Mohan S. Kankanhalli, and Harold Soh. Embedding symbolic knowledge into deep networks. In Hanna M. Wallach, Hugo

Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4235–4245, 2019.

[208] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR, 2018.

[209] Bishan Yang, Wen-Tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015*, 2015.

[210] Dounia Yedjour and Abdelkader Benyettou. Symbolic interpretation of artificial neural networks based on multiobjective genetic algorithms and association rules mining. *Appl. Soft Comput.*, 72:177–188, 2018.

[211] Jianbo Yu and Guoliang Liu. Extracting and inserting knowledge into stacked denoising auto-encoders. *Neural Networks*, 137:31–42, 2021.

[212] Yufei Yuan and Huijun Zhuang. A genetic algorithm for generating fuzzy classification rules. *Fuzzy Sets Syst.*, 84(1):1–19, 1996.

[213] Dengsheng Zhang. *Wavelet Transform*, pages 35–44. Springer International Publishing, Cham, 2019.

[214] Ying Zhang, Hongye Su, Tao Jia, and Jian Chu. Rule extraction from trained support vector machines. In Tu Bao Ho, David Wai-Lok Cheung, and Huan Liu, editors, *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*, volume 3518 of *Lecture Notes in Computer Science*, pages 61–70. Springer, 2005.

[215] Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the 34th Conference on Artificial Intelligence (AAAI), The 32nd Innovative Applications of Artificial Intelligence Conference (IAAI), The 10th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI), New York, NY, USA, February 7-12, 2020*, pages 3065–3072. AAAI Press, 2020.

[216] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: enhanced language representation with informative entities. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL), Florence, Italy,*

July 28 - August 2, 2019, pages 1441–1451. Association for Computational Linguistics, 2019.

[217] Hao Zhou, Tom Young, Minlie Huang, Haizhou Zhao, Jingfang Xu, and Xiaoyan Zhu. Commonsense knowledge aware conversation generation with graph attention. In Jérôme Lang, editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden, July 13-19, 2018*, pages 4623–4629. ijcai.org, 2018.

[218] Zhi-Hua Zhou, Shi-Fu Chen, and Zhaoqian Chen. A statistics based approach for extracting priority rules from trained neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000, Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, July 24-27, 2000, Volume 3*, pages 401–406. IEEE Computer Society, 2000.

[219] Zhi-Hua Zhou, Yuan Jiang, and Shi-Fu Chen. Extracting symbolic rules from trained neural network ensembles. *AI Commun.*, 16(1):3–15, 2003.

[220] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. DeepRED – Rule extraction from deep neural networks. In Toon Calders, Michelangelo Ceci, and Donato Malerba, editors, *Discovery Science - 19th International Conference, DS 2016, Bari, Italy, October 19-21, 2016, Proceedings*, volume 9956 of *Lecture Notes in Computer Science*, pages 457–473, 2016.