University of Padova

Department of Mathematics

Master Thesis in Data Science

Towards Explainability in Knowledge Enhanced Neural Networks

Supervisor
Luciano Serafini
University of Padova

Co-supervisor
Alessandro Daniele
Fondazione Bruno Kessler

Master Candidate
Riccardo Mazzieri

Academic Year
2020-2021

Dedication.

Abstract

This is the abstract of the thesis.

Contents

Listing of figures

Listing of tables

Listing of acronyms

EOA	Example Of Abbreviation
AEOA	Another Example Of Abbreviation

Contents

1 Introduction

This is the introduction.

The thesis is organized as follows. Some useful examples are detailed in Chapter 2. Concluding remarks are reported in Chapter 4.

2 Explainability vs Interpretability

Intepretability is less strong than Explainability. In fact... cfr 2. LMAO prova

2.1 Section 1 of chapter 1

Text of Section 2.1, in Chapter2. A figure is reported in Figure2.1.

Figure 2.1: Example of caption.

2.2 Section 2 of chapter 1

2.2.1 Subsection of section 2

This is a subsection. An example of table is in Table 2.1.

field1	field2
value1	value2

Table 2.1: Example

2.3 Section 3 of chapter 1

This is an example of equation: Equation 2.1.

$$y = \sum_{i=0}^{N-1} a_i x + b_i. \tag{2.1}$$

2.4 Section 4 of chapter 1

Some examples of references. This is a book reference[?]. This is an article reference[?]. This is a conference reference[?]. This is a reference for an online resource [?]. This is a reference for a standard[?].

3 Knowledge Enhanced Neural Networks

Knowledge Enhanced Neural Network (KENN) is a special type of model; more specifically, is a residual layer designed to by attached at the end of a standard Neural Network (NN), in order to boost its predictive performances via the addition of a Prior Knowledge, in the form of First Order Logic (FOL) clauses.

3.1 Theoretical Framework

We present here the theoretical framework behind KENN. The first step will be to rigorously define the symbolic language and how to link it with the theoretical framework of NNs, which consists in defining a semantic for the language. Next, we will describe the process with which the truth value of a clause can be increased, and how to integrate this method inside NNs.

3.1.1 Prior Knowledge and the semantic of $\mathcal{L}$

Definition 3.1.1 (Prior Knowledge). Collection of formulas of a function-free FO language $\mathcal{L}$ whose signature is defined with a set of constants $\mathcal{C} = \{a_1, \dots, a_l\}$ and a set of predicates $\mathcal{P} = \{p_1, \dots, p_q\}$. Each predicate can be applied to a specific number of constants n, which we will define as the *arity* of the predicate.

Definition 3.1.2 (Clause). A clause is defined to be of the following form:

$$c := \bigvee_{i=1}^{k} l_i, \quad l_i \neq l_j \quad \forall i \neq j. \tag{3.1}$$

where l_i is a literal, i.e. a formula constituted only by a n-ary predicate, or its negation. Also clauses have an arity, which is by definition the maximum arity of the predicates that constitute it.

One example of a clause could be the following:

$$c(x, y) = \neg Smoker(x) \vee \neg Friends(x, y) \vee Smoker(y) \tag{3.2}$$

which is equivalent to the clause $Smoker(x) \wedge Friends(x, y) \Rightarrow Smoker(y)$, but expressed as a disjunction of literals. Such a clause is constituted by two predicates: $Smoker(x)$, a unary predicate expressing the statement "*x is a smoker*", and $Friends(x, y)$, a binary predicate which expresses the statement "*x and y are friends*". Therefore, this clause expresses the rule "*if x is a smoker and x and y are friends, than also y is a smoker*". Note that the variables x and y are supposed to be universally quantified, since our aim is to express general knowledge. We now give another definition:

Definition 3.1.3 (Grounding of a clause). The grounding of an n-ary clause c, denoted as $c[x_1/k_1, \ldots, x_n/k_n]$, is the clause obtained by substituting k_i to $x_i, \forall i = 1, \ldots, n$.

Going back to the example of before, assume that a and b are two specific persons. Then, the grounding of clause (3.2) will be

$$c(x/a, y/b) = c(a, b) = \neg Smoker(a) \vee \neg Friends(a, b) \vee Smoker(b).$$

The next step is to build a semantic for the formal language $\mathcal{L}$, that is, how to interpret the symbols that we are working with. In practice, this will consist on defining a way to map constants towards a domain, and predicates to functions that go from such domain to a truth value. To clarify, consider the following example: let a be a constant and let P be a predicate, such that $P(x)$ expresses the statement "*x is a prime number*". In this case, there is a natural way to define an interpretation for our symbols, that is to map constants to the domain of natural numbers and to map P to the function $f : \mathbb{N} \longrightarrow \{0, 1\}$, where $f(n) = 1$ if n is prime, and 0 otherwise. Now, we define the semantic of $\mathcal{L}$.

Definition 3.1.4 (Semantic of $\mathcal{L}$). The semantic of $\mathcal{L}$ is defined by means of a pair of functions $(\mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{P}})$, that, together, define an *interpretation* for the symbols of our language and are defined as follows:

$$\begin{aligned} \mathcal{I}_{\mathcal{C}} : \mathcal{C} &\longrightarrow \mathbb{R}^l & \qquad \mathcal{I}_{\mathcal{P}} : \mathcal{P} &\longrightarrow \left(\mathbb{R}^{nl} \to [0,1]\right) \\ c &\longmapsto x, & P &\longmapsto f \end{aligned} \tag{3.3}$$

Where n is the arity of the predicate P and f is a function that takes in input the interpretations of n constant symbols, $\mathcal{I}_C(c_1), \ldots, \mathcal{I}_C(c_n)$ and returns the truth value of $P(c_1, \ldots, c_n)$. Note that, to make the notation lighter, we will omit the subscript when it's clear whether the argument of the interpretation is a literal or a constant term.

One could already see an analogy with the theoretical setup of NNs. In fact, each constant symbol c is mapped to a l-dimensional real vector, which can be seen as the feature vector characterizing the real world object identified by c. Another important detail is that the truth value of each literal, in our setup, is not determined by a hard assignment of 0 or 1, but is represented by a real number in the interval $[0, 1]$. This is a crucial point: indeed, the truth value in our semantic is trying to represent predictions by a NN, which are always expressed in terms of probability. The natural consequence of this choice is that, from this point on, we will have to rely on the rules of Fuzzy Logic, which is a generalization of the standard Boolean logic where the truth value of variables can take the value of any real number between 0 and 1.

3.1.2 T-Conorm Functions

By defining a semantic for $\mathcal{L}$, we can now give an interpretation for constants and predicates. The next step is to find a way to interpret clauses, or, more specifically, a way to determine the truth value of a grounded clause. We saw that, by definition, a clause is a disjunction of literals: this means that we only need a way to define the interpretation of a negated predicate and of the disjunction of two predicates. As stated above, since we are allowing truth values in the range $[0, 1]$, we will need to use the rules by Fuzzy Logic. For computing the truth value of a negated predicate, the standard way in Fuzzy Logic is to use the Lukasiewicz Negation.

Definition 3.1.5 (Lukasiewicz Negation). If $P \in \mathcal{P}$ is a predicate, then:

$$\mathcal{I}(\neg P) = 1 - \mathcal{I}(P)^*$$

*Writing $1 - \mathcal{I}(P)$ is a slight abuse of notation since $\mathcal{I}(P)$ is a function (or is it?).

So for example if the truth value of a predicate is $\mathcal{I}(P)(x) = 0.8$, the truth value of its negated copy would be $\mathcal{I}(\neg P)(x) = 0.2$. It is worth noting that this definition is equivalent to the Boolean negation when $\mathcal{I}(P) = 0$ or $\mathcal{I}(P) = 1$.

With this tool we are now able to compute the truth value of any literal. There remains to see how to define the interpretation of a disjunction of literals. To do this, we introduce the concept of T-Conorm functions.

Definition 3.1.6 (T-Conorm). A T-Conorm is a function $\perp \colon [0,1]^2 \to [0,1]$ that satisfies the following properties:

1. $\perp(a,b) = \perp(b,a)$
2. $\perp(a,b) \leq \perp(c,d)$ if $a \leq c$ and $b \leq d$
3. $\perp(a, \perp(b,c)) = \perp(\perp(a,b), c)$
4. $\perp(a,0) = a$

By definition, $\perp$ takes values in $[0,1]^2$, but can be easily extended to $[0,1]^n$ for any n, by defining:

$$\perp(a_1, \dots, a_n) := \perp(\perp(a_1, \perp(a_2, \cdots \perp(a_{n-1}, a_n))))$$

In Fuzzy Logic, T-Conorm functions are used to represent the concept of logical disjunction, and will be the tool employed to represent the interpretation of a disjunction of literals. Specifically:

$$\mathcal{I}(l_1 \vee \cdots \vee l_n) = \perp(\mathcal{I}(l_1), \dots, \mathcal{I}(l_n)) \tag{3.4}$$

It is also worth specifying that $\mathcal{I}(l_1 \vee \cdots \vee l_n)$ will be a function from $\mathbb{R}^{nl}$ to $[0,1]$, where n is the arity of the clause $c := \bigvee_{i=1}^{k} l_i$. With the given definitions, we have all that is needed to compute the truth value of any grounded clause. From a practical point of view, the only remaining step would be to choose a specific T-Conorm function. KENN uses the Gödel T-Conorm function, which is also known as the Maximum T-Conorm and is defined as

$$\perp_{max}(a,b) = \max\{a,b\},$$

which, as above, can be extended like follows:

$$\perp_{max}(t) = \max_{i=1,\dots,l} t_i, \quad \forall t \in \mathbb{R}^l.$$

We are now finally ready to fully understand how this theoretical framework is able to describe the predictions of a NN. Suppose that we have a dataset $\mathcal{X} = \{x_1, \dots, x_n\}, x_i \in \mathbb{R}^l$, where each x_i belongs to one or more classes $(P_1, \dots, P_m)$. The task in which the NN must learn to classify each input into one or more output classes is known in Machine Learning as a multilabel classification problem. To tackle this kind of task, a NN architecture will present, in the last layer, m output units, each of which will be finally subject to a sigmoidal activation function. After training, the NN will have learned to approximate a function $h(x_i) = y_i \in \mathbb{R}^m$, where $(y_i)_j = \mathbb{P}(x_i \text{ belongs to class } j)$. Now, if we consider:

1. $\mathcal{P} = \{P_1, \dots, P_m\}$ to be predicates defined as $P_i(x) =$ "x belongs to class P_i";

2. $\{x_1, \dots, x_n\}$ to be the interpretations of the constants $\mathcal{C} = \{c_1, \dots, c_n\}$, which represent the real-world objects of our dataset,

it is clear that the entries of y_i can be seen as truth values of the predicates $\{P_1, \dots, P_m\}$. More formally:

$$(y_i)_j = \mathcal{I}_{NN}(P_j)(x_i), \quad \forall i = 1, \dots, n, \forall j = 1, \dots, m. \tag{3.5}$$

Hence, the whole NN defines an interpretation for each predicate P_i, which we denoted as $\mathcal{I}_{NN}$. Therefore, given a clause $c := \bigvee_{i=1}^{k} l_i$ and given and $\{x_1, \dots, x_d\}$ a collection of feature vectors (where d is the arity of c), then the truth value of the grounded clause predicted by the NN will be $\perp (y_c)(x_1, \dots, x_k)$, where:

$$y_c \in \mathbb{R}^k, \quad `(y_c)_i = \begin{cases} \mathcal{I}(l_i) \text{ if } l_i \text{ is not a negated predicate} \\ 1 - \mathcal{I}(l_i) \text{ otherwise.} \end{cases} \tag{3.6}$$

The intuition behind KENN is very simple: given y the vector of predictions by the NN, a new layer is added at its end with the aim to modify y and obtain a new vector of predictions y', of the form $y' = y + \delta$, such that y' improves the truth value of each clause present in the base knowledge and, at the same time, keeps the quantity $\|y' - y\|_2$ minimal. It is worth noticing that this new layer introduced by KENN, called Knowledge Enhancer (KE), is a kind of residual layer, since it learns to represent the quantity $\delta = y' - y$.

3.1.3 T-Conorm Boost Functions

The next problem is to understand how to improve the truth value of a single clause. Since this truth value is represented by a T-Conorm function, this involves finding a way to let the value of $\perp(y)$ rise by manipulating the value of y. To do this, we define a new class of functions.

Definition 3.1.7 (T-Conorm Boost Function (TBF)). A function $\delta : [0,1]^n \to [0,1]^n$ is a T-Conorm Boost Function (TBF) if:

$$0 \leq t_i + \delta(t)_i \leq 1 \quad \forall n \in \mathbb{N} \quad \forall \mathbf{t} \in [0,1]^n.$$

Let Δ denote the set of all TBFs.

From the definition follows a simple but essential result.

Lemma 3.1.1. Given $\perp$ any T-Conorm and $\delta \in \Delta$, it holds that:

$$\perp(t) \leq \perp(t + \delta(t)).$$

Proof. By definition of TBF, $\delta(t)_i \geq 0$ and also $t_i \geq 0$. This implies that

$$t_i \leq t_i + \delta(t)_i, \quad \forall i = 1, \dots, n.$$

By the monotonicity of T-Conorms, it follows that $\perp(t) \leq \perp(t + \delta(t))$. $\square$

The purpose of such TBF δ is to update the NN predictions $y \in \mathbb{R}^m$ to a new vector $y' = y + \delta(y)$, in such a way that the truth value of each clause increases. The problem is now how to choose such a TBF. It is clear that not all the $\delta \in \Delta$ would be useful: for example, one could choose the function $\delta(y)_i = 1 - y_i, \quad \forall i = 1, \dots, n$. In this way we would obtain a truth value of 1 for any clause, independently of y. This of course would be pointless, and would render the predictions of the base NN useless. For this reason another requirement for y' is needed. Specifically, as we already mentioned, KENN is built in such a way that the learnt δ improves the T-Conorm value in a minimal way. To be more rigorous, we will now formally define the concept of a minimal TBF.

Definition 3.1.8 (Minimal TBF). A function $\delta \in \Delta$ is minimal with respect to a norm $||\cdot||$ and a t-conorm $\perp$ if and only if:

$$\|\delta'(t)\| < \|\delta(t)\| \Rightarrow \perp(t + \delta'(t)) < \perp(t + \delta(t)), \quad \forall \delta' \in \Delta \quad \forall n \in \mathbb{N} \quad \forall t \in [0,1]^n$$

3.2 KENN Architecture

4 Conclusion

The conclusion goes here.

Acknowledgments

This is the acknowledgments section.