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

NN ..... Neural Network

KENN ..... Knowledge Enhanced Neural Network

TBF ..... *t*-conorm Boost Function

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 language semantic

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 $\mathcal{C} = \{a_1, \ldots, a_l\}$ and a set of predicates $\mathcal{P} = \{p_1, \ldots, p_q\}$. 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

With our definition of 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 of 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:

$$\mathcal{I}(\neg P) = 1 - \mathcal{I}(P)^* \tag{3.4}$$

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:

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

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:

$$\mathcal{I}(l_1 \vee \cdots \vee l_n) = \perp(\mathcal{I}(l_1), \ldots, \mathcal{I}(l_n)). \tag{3.5}$$

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

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

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

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:

$$(y_i)_j = \mathcal{I}_{NN}(P_j)(x_i), \quad \forall i = 1, \dots, n, \forall j = 1, \dots, m. \tag{3.6}$$

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:

$$y_c \in \mathbb{R}^k, \quad (y_c)_i = \begin{cases} \mathcal{I}(l_i) \text{ if } l_i \text{ is not a negated predicate} \\ 1 - \mathcal{I}(l_i) \text{ otherwise.} \end{cases} \tag{3.7}$$

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

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

By the monotonicity of t-conorms, it follows that $\perp(t) \leq \perp(t + \delta(t))$. □

The purpose of such TBF δ is to update the NN predictions $y \in \mathbb{R}^m$ to a new vector 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 $\delta(y)_i = 1 - y_i, \quad \forall i = 1, \ldots, n$. In this way, we would obtain an updated 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:

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

As mentioned above, KENN works with the Gödel t-conorm function and the L_p norm $\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$. The next step at this point is to find such a minimal TBF. In the following result, we present a possible form that a minimal TBF can assume.

Theorem 3.1.2. For any function $f : \mathbb{R}^n \to \mathbb{R}$ we define $\delta^f : \mathbb{R}^n \to \mathbb{R}^n$ as

$$\delta^f(t)_i = \begin{cases} f(t) & \text{if } i = \arg\max_{j=1}^{n} t_j \\ 0 & \text{otherwise} \end{cases}$$

Let $f : [0,1]^n \to [0,1]$ satisfying $0 \leq f(t) \leq 1 - \max_{j=1}^{n} t_j$. Then, δ^f is a minimal TBF for the Gödel t-conorm function and the L_p norm.

Proof. δ^f is a TBF. Indeed $\delta^f(t) \geq 0$ and $0 \leq t_i + \delta^f(t_i) \leq 1$ because $f(t) \leq 1 - \max_j t_j$. Therefore we only need to prove that δ^f is minimal. Take $\delta \in \Delta$, with $\|f(t)\|_p < \|\delta^f(t)\|_p$. We have to show that

$$\perp (t + \delta(t)) \leq \perp \left(t + \delta^f(t)\right).$$

Now define $j = \arg\max_k (t_k + \delta(t)_k)$. By definition of the Gödel t-conorm we can immediately derive that:

$$\perp (t + \delta(t)) = t_j + \delta(t)_j. \tag{3.8}$$

Now, defining $i = \arg\max_k t_k$, using the same reasoning and exploiting the definition of δ^f it follows that:

$$\perp (t + \delta^f(t)) = t_i + f(t). \tag{3.9}$$

By combining (3.8) and (3.9) and noting that by definition $t_i \geq t_j$, the last step is to prove that $f(t) > \delta(t)_j$. To do this we exploit the definition of L_p norm as follows:

$$\delta(t)_j = (|\delta(t)_j|^p)^{\frac{1}{p}} \leq \left(\sum_{k=1}^{n} |\delta(t)_k|^p\right)^{\frac{1}{p}} = \|\delta(t)\|_p < \|\delta^f(t)\|_p = f(t).$$

Where the last inequality follows from the definition of $\delta^f(t)$.

□

This makes sense even from an intuitive point of view: since $\perp(a) = \max_i a_i$, the only way to increase $\perp(a)$ is to let $\max_i a_i$ increase, without modifying the rest of the $a_j, j \neq i$.

3.1.4 Applying TBFs to preactivations

There is a problem with the definition of δ^f: there is a specific constraint $f(t) \leq 1 - \max_i t_i$ that limits the number of candidates for f. Indeed, this is imposed to ensure that the final output $y' = y + \delta^f(y)$ will be in $[0, 1]$. There is a natural way to solve this impracticality: since we are assuming a multilabel classification scenario, the final m output units of the NN will pass through a sigmoidal activation function. More specifically, y_i will be of the form:

$$y_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}, \forall i = 1, \ldots, m.$$

For this reason, KENN exploits the fact that $\sigma : \mathbb{R} \to [0, 1]$ by applying the TBF directly on the preactivations $z \in \mathbb{R}^m$. In fact, it is clear from an intuitive point of view that one can apply any delta to the preactivations vector, and at the same time always be sure that the final output y will be in $[0, 1]$. In this way, the constraint on f is no longer needed. The next theorem proves formally that applying a minimal TBF on the preactivations is equivalent to applying a minimal TBF on the output of the NN.

Theorem 3.1.3. For all $f : \mathbb{R}^n \to \mathbb{R}$, the function:

$$\delta^g(t) = \sigma(z + \delta^f(z)) - \sigma(z)$$

is a minimal TBF under the Gödel t-conorm and the L_p norm.

Proof. (Still to write) □

Note that $\delta^g(t)$ is not directly used in KENN but it's indirectly induced by using $\delta^f(z)$ on the preactivations. Applying the TBF directly on the preactivations has also another remarkable advantage. Indeed, it is known that it is possible to interpret the value of the preactivation of the i-th output neuron as the "confidence" of the NN that the current feature vector is to be classified in the i-th class. This "confidence" is not yet a probability, but a generic scalar value $z \in \mathbb{R}$; it will become a probability when transformed with the sigmoid activation function: $\sigma(z) \in [0, 1]$. More specifically we know that:

- $z \gg 0$ means high confidence of being classified in the i-th class. This follows from the fact that $\lim_{z \to +\infty} \sigma(z) = 1$;

- $z \ll 0$ means high confidence of *not* being classified in the i-th class. This follows from the fact that $\lim_{z \to -\infty} \sigma(z) = 0$;

- $z \approx 0$ corresponds to a highly uncertain decision. This follows from the fact that $\sigma(z) \approx 0.5$ if $z \approx 0$.

By observing the shape of the sigmoid activation function we can notice that when $|z| \gg 0$ (high confidence in the NN predictions), even large deltas on the preactivations produce very small changes. More rigorously, $\lim_{|z| \to \infty} \frac{d}{dz}\sigma(z) = 0$. On the contrary, when $z \approx 0$, even small deltas on the preactivations produce high modification at the activation level. This will result in the following behavior: if the NN is highly confident of its decision, then logical rules will not modify too much the result of the NN predictions. On the contrary, in the cases where the NN is uncertain of its decision, our base knowledge will intervene and give higher modifications on the final predictions. This conforms to the intuition that KENN should produce minimal changes in the original predictions. These key concepts are further illustrated in Figure 3.1.

As we already mentioned, the minimal TBF directly modeled by KENN is the one we called $\delta^f(z)$. From its definition, we know that the magnitude of the produced delta is determined by the definition of f. One of the most important features of KENN is that, by design, it learns to give the proper *importance* to each clause in the base knowledge: this precise feature of the model gives also a way to find such function f. Specifically, for each clause c a learnable parameter w_c is defined so that the produced delta for c is:

$$\delta^{w_c}(z)_i = \begin{cases} w_c & \text{if } i = \arg\max_{j=1}^{n} z_j \\ 0 & \text{otherwise.} \end{cases}$$

From this definition it's now clear that the function f we were looking for is not actually the same for all the clauses in the base knowledge, but it is defined for each different clause and it's equal to the constant function $f_c(z) = w_c, \quad w_c \in [0, \infty]$. There is one last problem: while it's true that the function δ^{w_c} is a minimal TBF, the implementation of this kind of functions inside a NN is unfeasible since they are not differentiable. For this reason KENN uses a soft approximation of δ^{w_c}, defined as:

$$\delta_s^{w_c}(z)_i = w_c \cdot \text{softmax}(z)_i = w_c \cdot \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}. \tag{3.10}$$

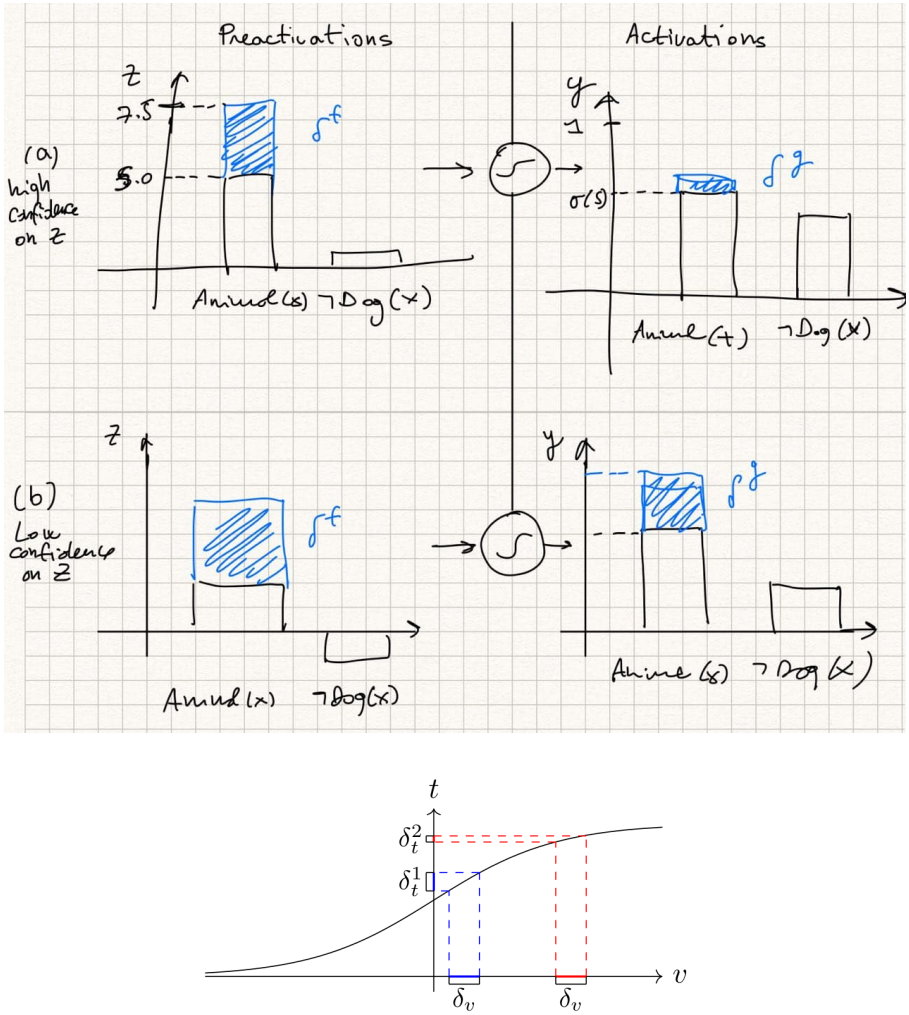

Figure 3.1: This image illustrates the actual deltas produced by KENN, which are the δ^f, opposed to the actual delta produced on the activations, which is δ^g. As we said, δ^g is not produced directly by the model but it is indirectly *induced* by the application of δ^f on the preactivations. This also illustrates how, thanks to the shape of the sigmoid activation function, the same delta on the preactivation produces a different delta at the activations level: the closer the preactivations to zero, the highest the modification on the final predictions.

There are still, however, some steps to describe in order to fully understand how KENN produces a vector of deltas. Recall our notation: we defined with $y \in \mathbb{R}^m$ the vector of predictions from the NN. Specifically, we now define with y_A the truth value relative to A, where A is a generic predicate of our language. We also define $z_A = \sigma^{-1}(y_A)$. Now, we note that equation (3.10), tells us that the produced δ is always m-dimensional, where m is the number of output classes. This however is not desirable: in fact, in general, not all the clauses in our base knowledge contain all the predicates of our language. For example, given $\mathcal{P} = \{A, B, C\}$, the

clause $c = A \vee \neg B$ contains only two of the three predicates in the language. Therefore, we would like this specific clause to not modify in any way z_C. Another problem is that we don't know how to express the preactivation of a negated literal, i.e. we don't know how to derive $z_{\neg A}$ from z_A. In fact, recall that in equation (3.4) we defined the interpretation of a negated predicate, where we knew that the truth values were well defined in the interval $[0, 1]$. Now we are dealing with preactivations, which cannot be considered truth values in the Fuzzy Logic theoretical framework. However, this problem can be easily solved by exploiting the following property of the sigmoid activation function:

$$1 - \sigma(x) = \sigma(-x).$$

Now it's easy to see that, since $y_{\neg A} = 1 - y_A$, we can define:

$$z_{\neg A} = -z_A.$$

Notice that we are not introducing any new concepts: instead we are just redefining quantities that were already mentioned at the activation level, to the preactivation level. We finally define $z_c = (z_{l_1}, \ldots, z_{l_k})$ for every clause $c := \bigvee_{i=1}^{k} l_i$ of the knowledge. We refer to the process of transforming from z to z_c as the *selection* step. This new vector contains only the preactivations of literals present in c, and is the one that we actually want to use to produce the delta relative to clause c. Now, let $\mathcal{K}$ be the set of clauses in our knowledge, and $\{w_c\}_{c \in \mathcal{K}}$ their corresponding weights. For every clause $c \in \mathcal{K}$ we want to obtain a new delta, namely $\delta^c \in \mathbb{R}^m$, which contains one value for each predicate in the clause and is defined as follows:

$$\delta^c_A = \begin{cases} \delta_s^{w_c}(z_c)_A & \text{if } A \in c \\ -\delta_s^{w_c}(z_c)_{\neg A} & \text{if } \neg A \in c \\ 0 & \text{Otherwise} \end{cases}, \quad \forall A \in \mathcal{K} \tag{3.11}$$

This newly defined delta, δ^c, will be the delta obtained from clause c and will be summed to z to obtain the updated prediction. More specifically:

$$y' = \sigma(z + \delta^c)$$

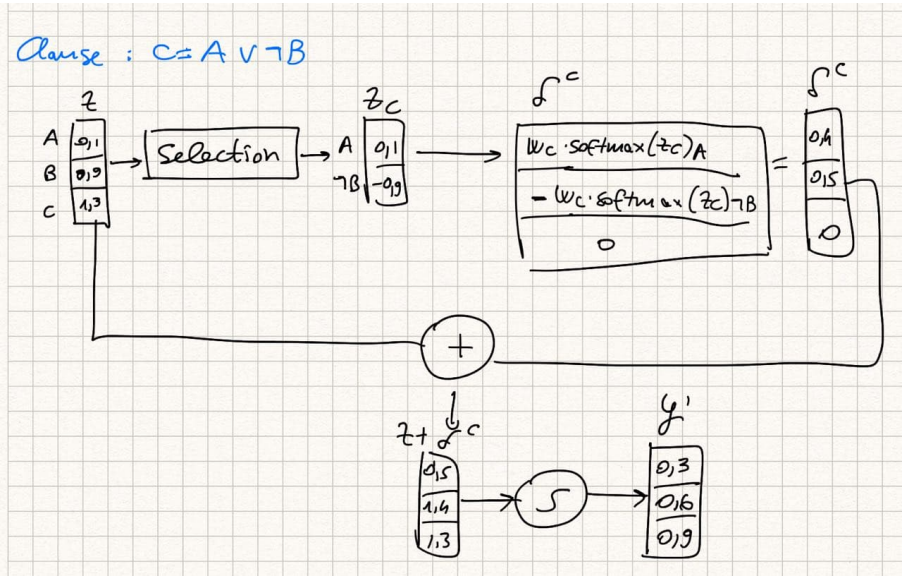

Figure 3.2: Summary of all the steps needed to produce δ^c, the vector of deltas derived from a single clause. We refer to this process as *clause enhancement*.

3.1.5 Increasing the satisfaction of the Knowledge

In the previous section we found out how KENN produces a vector of changes δ to be applied to the original NN predictions, but only considering a single clause. In the cases where the knowledge is constituted only by a single clause, we would already know how KENN works, but in real applications a higher number of logical rules will be desirable. Therefore, the next and final problem is to understand how KENN takes all the deltas from all the clauses and produces a single vector of changes. This particular step of aggregation is critical, as it constitutes one of the best features of KENN, but at the same time one of its bigger inaccuracies. This is because, to aggregate the contributions from all the clauses $c \in \mathcal{K}$, KENN just sums the contributions. Specifically, the final prediction is defined as follows:

$$y' = \sigma(z + \sum_{c \in \mathcal{K}} \delta^c).$$

This particular choice makes KENN really fast at inference and learning time, increasing scalability. At the same time, though, this makes the risk of inconsistencies higher. For example, the same predicate can appear negated in one clause, and not negated in another clause: in this way the delta for the first one will be negative while it will be positive for the second one. In this way, the two delta may cancel out rendering KENN less useful.

3.2 KENN Architecture

4
Conclusion

The conclusion goes here.

Acknowledgments

This is the acknowledgments section.