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

ML	Machine Learning
XAI	Explainable Artificial Intelligence
NN	Neural Network
KENN	Knowledge Enhanced Neural Network
TBF	t-conorm Boost Function
CE	Clause Enhancer

1
Introduction

This work of Thesis was conducted during my stage at Fondazione Bruno Kessler in Trento, Italy. During this experience, I worked alongside professor Luciano Serafini and researcher Alessandro Daniele in further experiments on Knowledge Enhanced Neural Networks, a special kind of Neural Network layer designed to inject logical knowledge inside a Neural Network in order to improve its predicting capabilities. This Thesis revolves around the idea of explainability... The Thesis is organized as follows: in Chapter 2 we will discuss the concept of explainability in Machine Learning and the last developments in the field of Explainable Artificial Intelligence. In Chapter 3 we will then see in detail the theory and architecture behind KENN, together with experimental results. Finally, in (), we will see how KENN can be considered an interpretable and explainable model, we will express some results and discuss about future work.

2 Explainability in Machine Learning

In the last years, thanks to the availability of always larger datasets and due to the always growing computing power of modern GPUs, a lot of artificial intelligence (AI) and deep learning (DL) systems have reached remarkable results in terms of performance on a wide variety of tasks, in some cases largely overcoming human capabilities. Examples are the fields of computer vision, speech recognition or machine translation, which almost always fall into the domain of supervised learning. DL, being at the moment the most successful ML approach in supervised learning, has been criticized in [3] for its lack of transparency: in fact, DL systems have millions or even billions of parameters, which are not characterized in human interpretable ways, but only in terms of their position in the network topology. This results in opaque models, to the point that such systems are currently treated almost always as black boxes. DL systems also present other critical issues: in [4] the authors made a neural network misclassify an image by applying an hardly perceptible perturbation, found by maximizing the network's prediction error. Such adversarial examples have also been found to be somewhat universal and not just the result of overfitting. This poses serious doubts about the ability of NN to learn general representations: indeed, if such networks can generalize well, how can they be confused by nearly indistinguishable images? Similarly, authors in [5] show how deep neural networks are easily fooled into misclassifying inputs with no resemblance to the true category. Adversarial examples are not confined to the field of computer vision: natural language networks can also be fooled as shown in [6]. Furthermore, it has been found that in several applications, DL systems present strong biasedness. One example is reported in [7], where the authors show how word embeddings trained on Google News articles exhibit strong female/male gender stereotypes due to biases in the training data. Susceptibility to unintuitive errors remains therefore a pervasive problem in DL and no robust solution has been found for them so far. Such issues contribute to generate mistrust, and threaten to slow down or even hinder the prevailance of AI in some applications, due to the high potential of unexpected behavior and lack of verifiability of solutions. In the light of such problems, explainable artificial intelligence (XAI) has become an area of interest in the research community: it tackles the important problem that complex machines and algorithms often cannot provide insights into their behavior and thought processes. The need for XAI is now even more urgent: the renewed EU General Data Protection Regulation (GDPR) could require AI providers to provide users with explanations of the results of ML systems based on their personal data. This clearly affects the industry in a huge way: indeed, the GDPR may hinder or even prohibit the use of "black box" models which don't offer explanations for their decisions, when based on users' personal data (think for example to recommender systems). This is also referred to as the "right to explanation". The need for XAI has been also expressed by the statement on algorithmic transparency and accountability released by the Association for Computing Machinery [8], and by the XAI program launched by DARPA in 2017 [9].

Even though the general aim for XAI is well understood as the achievement of *interpretability*, or *explainability*, for ML models, few articulate precisely what those terms mean or why they are important. Therefore, the first steps of this chapter will be to define What is explainability, Who needs it, When it is needed and Why.

2.1 Why do we need explainability

Before determining a good definition of *explainability* or *interpretability*, we must have a good understanding of what the real world objectives of XAI research are. Consider a supervised learning scenario: a lot of evaluation metrics are used to assess the quality of a model, accuracy probably being the most common one. The computation of such metrics require the presence of model predictions, together with ground truth labels, in order to produce a score. Unfortunately, this evaluation framework is failing to satisfy the request for interpretability: therefore we should investigate what are those other desiderata which are being demanded, and what are the circumstances in which they are sought. Lipton [10] suggests that this kind of situations arise "when our real world objectives are difficult to encode as simple real-valued functions": in this sense, *interpretations* are useful to achieve objectives which are important for us, but which we struggle to model in a formal way. Other mentioned motivating aspects are causality, transferability, informativeness and fair and ethical decision making. The authors in [11] refer to this same concept as *incompleteness* in the problem formalization: in such situations, explanations are one of ways to ensure human intervention on such questions, where machines are still struggling to understand. Some examples of such scenarios would be:

- **Scientific Understanding**: humans learn about the world around them in the form of knowledge, which is still difficult to formalize in the same way in which it works inside our brains. For this reason, we might look for explanations from ML models, which in turn we can interpret and transform into human interpretable knowledge.
- **Safety**: in complex tasks, rigorous and complete testing is almost never feasible and it is not possible to formally model all the wrong or dangerous decisions that a model can make. For this reason, when decisions from a ML system can pose a threat to others, explanations can help humans to evaluate safety conditions and to boost trust towards AI.
- **Ethics**: for us humans, evaluating the fairness of a decision is often easy, in the same way in which we have a clear idea of how we would want our model to be ethical (for example, we could desire a "fair" classifier for loan approval). This kind of properties are not easy to encode in ML systems and at, the same time, biases in the data can often to unethical decisions if not treated properly.

Some papers [12, 13] also motivate the need for explainability and interpretability in light of the need for trust by domain experts: indeed trust is fundamental if one plans to act based on a prediction, therefore ML systems must be able to communicate with highly skilled human experts to leverage their expertise and share useful information or patterns from the data.

2.2 When do we need explainability

Explainability is important in a lot of domains, but not in all of them. There are applications, e.g. aircraft collision avidance, in which algorithms have been functioning from years without giving any explanations and without any human interaction. It is clear then that ML systems can be used without any need for interpretations in real world applications, in those cases where their raw performance in terms of accuracy suffices, or when the risk of error doesn't pose any serious threat to the end users. Therefore, domains that demand explainability are generally characterized by the critical nature of decisions which need to be made, where mistakes could have severe consequences. Authors in [14] provide a fairly exhaustive list of domains in need for explainability:

- Medical Domain/Health-Care:
- Judicial System
- Banking/Finance
- Bio-informatics
- Automobile Industry
- Marketing
- Election Campaigns
- Precision Agriculture
- Expert Systems for the Military
- Recommender Systems

2.3 What is explainability

Several research works attempt to describe rigorously the meaning of explainability in the contest of XAI. In the literature, such word is often used interchangeably or substituted by "intrepretability", even though some try to make a distinction. In [1], for example, the authors claim that explainability is a property of a model that implies interpretability, but not viceversa. More specifically, *interpretability* is described as the capability of a certain model to be described in a way that is understandable to humans; on the other hand, *explainability* is the property of a model to be able to summarize the reasons for their behavior. Explanations, according to the authors, can be evaluated in two ways: according to their *interpretability* (that is, its understandability by a human being) and their *completeness* (that is, the accuracy of the description). Under this definitions, the challenge of XAI is in creating explanations that are both interpretable and complete, even though such characteristics are often opposed to one another. These two features of explanations resemble two important properties of ML models

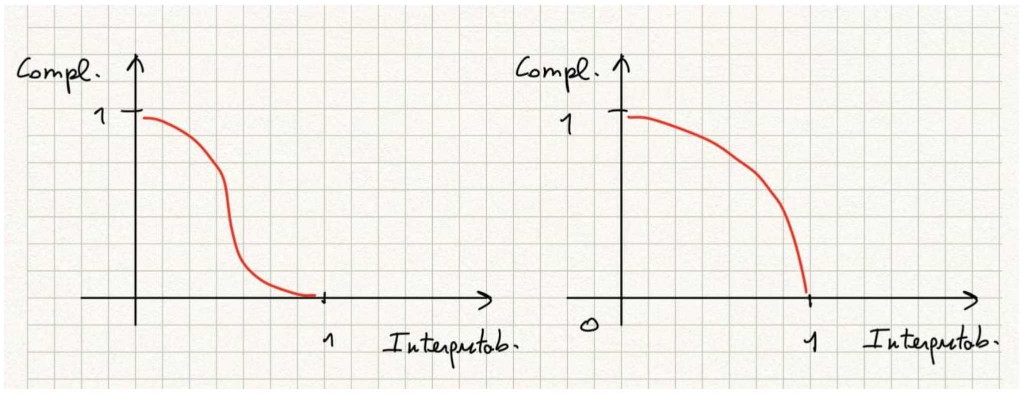

Figure 2.1: Example of two different explanations according to the definition given by [1]: each explanation allows for a tradeoff between completeness and interpretability. Explanation on the right should be evaluated better than the one on the left: indeed, for higher values of interpretability, the right one offers higher values of completeness.

and suggest a similitude: on one hand, the user would desire a simple model with few parameters and at the same time a model able to capture really well the structure of the training data. While both the properties are desirable, they are almost never achievable at the same time. Indeed, as we train simple models, we will probably underfit the data: in the same way, really easy explanations often fail to capture the complexities behind the internal workings of our algorithms. On the other hand, as we add parameters to our model and make it more complex, it will begin to better fit the data: in the same way a really complete explanation will describe accurately the operations of the systems, but will probably result more difficult to understand to a human being. This comparison suggests that, even for explanations, one should allow for a *tradeoff* between interpretability and completeness. The author suggests also that the explanation methods should be evaluated according to how such explanations behave on the curve from maximum interpretability to maximum completeness. Figure 2.1 gives a visualzation of this concept.

In [11] interpretability is defined as the ability to explain or to present in understandable terms to a human. According to [14], instead, *interpretability* is most often used in terms of comprehending how the prediction model works as a whole, while *explainability*, in contrast, is used to indicate the capability of models to give explanations about their decision, but keeping their black box nature. In the particular context of generating explanations for DL models, the authors of [15] define an explanation as the collection of features of an interpretable domain (i.e. pixel values on images), that have contributed for a given example to produce a decision (e.g. classification or regression). We can see that none of the aforementioned definitions are specific enough to enable one universal formalization: indeed they implicitly depend on the context or the aim of the research work. We will now try to give definitions that better summarize the mentioned ones.

Definition 2.3.1 (Interpretability)**.** We define *interpretability* in the context of supervised ML as the generic property of a model which makes its single components, as well as its functioning as a whole, understandable by a human being. Examples of interpretable models are simple linear regression or decision trees. Examples of not interpretable models are deep neural networks.

Definition 2.3.2 (Explainability)**.** We define *explainability* in the context of supervised ML as the generic property of a model which makes it able to explain, to a certain extent, its own reasons behind a certain decision.

In general, it is true that an explainable model is almost always interpretable, but the viceversa might not always be true. An exception to this rule, however, even if outside the field of ML, is the human brain: while we are able to give really detailed and motivated reasons behind our decision processes, our brain is not an interpretable model. Indeed, we don't know every single aspect of how our brains works, and yet we (often) trust the explanations that other human beings provide when asked why they took a certain decision. This offers an interesting point of view to the discussion: we should be careful not to trust certain explanations only by the fact that they look plausible and convincing. To this regard, Herman [16] warns us, making a clear distinction between descriptive and persuasive explanations: indeed implicit cognitive biases of the human brain could threaten transparency (for example, humans naturally tend to prefer simpler descriptions). To avoid falling in this problem, one should always keep in mind the tradeoff process between completeness and interpretability, mentioned above.

Even after an attempt of definitions, the meaning of interpretability, and explainability is still too generic and can be applied to any ML model. In fact, the volume of research in interpretability is quickly expanding, making the number of available methodologies continuously grow. Despite that, two main categories of approaches to interpretability and explainability are often distinguished [17, 10]: integrated interpretability, or *transparency* and *post-hoc* interpretability.

2.3.1 Transparency

Transparency is one of the properties that can enable interpretability and it implies some sort of understanding of the mechanism by which the model works. It can also be seen as the direct opposite to the concept of *black box*. Lipton [10] goes into even more details, by subdividing transparency in different levels:

1. **Simulatability**: it's the highest level of abstraction of the concept of transparency. Lipton refers to simulatability as the property of the model that makes it understandable by a person "at once". Specifically, this means that a human could, given the input data and all the necessary parameters, produce a prediction by making all the computations in a reasonable time.

2. **Decomposability**: it's the transparency considered at the level of the single components of the model. Specifically, one model can be considered decomposable if each part of the model (weights, modules, computations...) admits an intuitive explanation.

3. **Algorithmic Transparency**: this notion of transparency refers to the learning algorithm itself. For example, we know that in the case of linear regression the shape of the loss function is known, as well as an analytical form for the solution for the problem. This means a maximum degree of algorithmic transparency. On the other hand, modern deep learning lacks this notion of transparency: in fact, even if a lot of powerful optimization algorithms give empirically excellent results, there is no guarantee that those will work on any new problem. The same holds for the shape of the error function, which is almost never known.

It's interesting to notice that the human brain, as noted previously, doesn't exhibit any of those features. In fact, human thinking is not transparent to us and justifications in the form of explanations may differ from the actual decision mechanism.

2.3.2 Post-hoc interpretations

Post hoc interpretability is a different approach to interpretability: it refers to those methods with which we generate explanations from already trained models, without caring about their internal mechanisms. The advantage of this approach is that it does not impact performance of the model, which is treated as a black box. Unlike transparency, this kind of interpretability is the one that applies to humans. Lipton [10] summarizes post-hoc explainability methods into four main categories:

- **Text explanations** :This approach aims to mimic the way humans provide explanations to one another, that is, in text form. Example of such approaches: in reinforcement learning [18], in recommender systems: [19]...

- **Visualization**: Another popular approach to post-hoc interpretations is to provide visualizations in order to have a qualitative idea of what the model has learned. For example, when models learn embeddings in high dimensional vector spaces, one popular technique to visualize them is t-SNE [20], which provides 2D or 3D visualization of high dimensional data points, in such a way that nearby samples are likely to appear closer together. In the field of computer vision, several papers have investigated the internal representations of visual concepts in CNNs. In [2], the authors try to build "prototipes" of certain objects starting from already trained image classification models. Specifically, starting from a white noise image, they tweak it in such a way that the activation of a certain neuron (in this case, the output neuron corresponding to the selected object) is maximized. This process can be replicated also starting from an existing image, a process which produces really peculiar visual effects (see Figure 2.2). This process is also known as *Activation Maximization*: consider a deep NN classifier which maps an input tensor (in this case an image) x to a set of classes $\{\omega_i\}_{i=1}^m$. In a classification scenario, we know that the i-th output neuron encodes the modeled class probability $p\left(\omega_i|x\right)$. The basic idea is that the *prototype* x_i^*, representative of class ω_i can be found as follows:

$$x_i^* = \max_x \log p(\omega_i|x) - \lambda\|x\|^2.$$

  The proposed definition doesn't yield good results in practice: although producing strong class response, they often look unnatural. This problem is solved in several ways, for example by adding regularization via a data density model, or imposing prior constraints which are common in real images, such as high correlation among neighboring pixels. Again, a similar approach is found in [21], where the authors investigate the amount of information retained in the hidden representations of CNNs. They manage to reconstruct the original images with good accuracy even from high level representations by performing gradient descent on white noise inputs.

- **Local explanations** In [22, 15] saliency maps/relevance scores. [13] interpretable and faithful manner, by learning an interpretable model locally around the prediction... Probably also attention mechanisms... '

- **Explanation by Example**

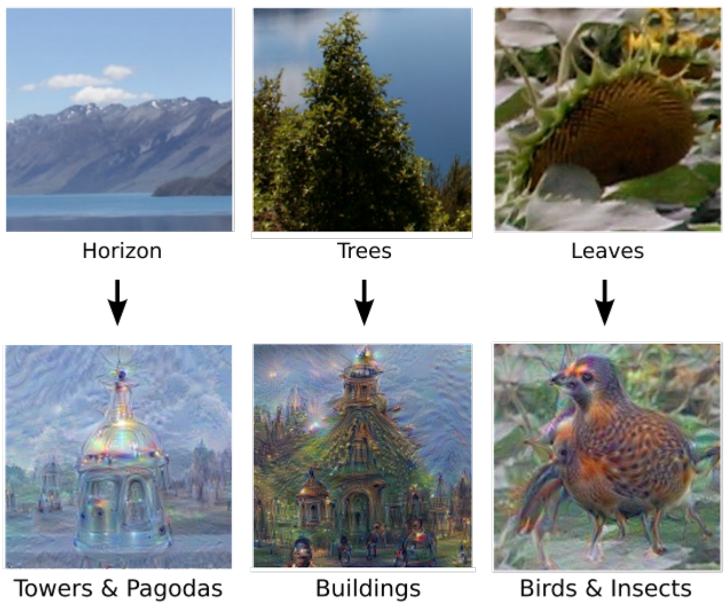

Figure 2.2: Process of image manipulation shown in [2]. From an intuitive point of view, starting from existing real images, the network is asked to enhance the features of the image that resemble the desired object (in this case, starting from an image of a tree, start looking for features that resemble buildings). This process is then repeated, creating a positive feedback loop. As a result, the features of the desired objects appear seemingly out of nowhere.

3
Knowledge Enhanced Neural Networks

Knowledge Enhanced Neural Networks (KENN) is a special type of Neural Network (NN) layer, designed for injecting logical knowledge into a pre-existing base NN. More specifically, it is a residual layer designed to be stacked after the last layer of a standard NN, in order to boost its predictive performances via the addition of a Prior Knowledge in the form of first order logic clauses. In this chapter we will describe the theory behind KENN, its architecture, and experimental results.

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, $\mathcal{I}_C(c_1), \dots, \mathcal{I}_C(c_n)$ and returns the truth value of $P(c_1, \dots, c_n)$. 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:

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

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:

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

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?).

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

which, as above, can be extended like follows:

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

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 $\mathcal{X} = \{x_1, \ldots, x_n\}, x_i \in \mathbb{R}^l$, where each x_i belongs to one or more classes $(P_1, \ldots, P_m)$. 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. $\mathcal{P} = \{P_1, \ldots, P_m\}$ to be predicates defined as $P_i(x) =$ "x belongs to class P_i";
2. $\{x_1, \ldots, x_n\}$ to be the interpretations of the constants $\mathcal{C} = \{c_1, \ldots, c_n\}$, 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 $\{P_1, \ldots, P_m\}$. More formally:

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

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 $\{x_1, \ldots, x_d\}$ 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 $\bot\,(y_c)(x_1, \ldots, x_k)$, 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:

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

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 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 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 $0 \le f(t) \le 1 - \max_{j=1}^n t_j$. 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 $\delta^f(t) \ge 0$ and $0 \le t_i + \delta^f(t_i) \le 1$ because $f(t) \le 1 - \max_j t_j$. 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

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

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 $t_i \ge t_j$, 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:

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

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 z is equivalent to applying a minimal TBF on the output of the NN y.

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

$$\delta^g(y) = \sigma(z + \delta^f(z)) - \sigma(z) \tag{3.10}$$

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

Proof. By definition we know that $z = \sigma^{-1}(y)$, hence we can rewrite equation (3.10) as:

$$y + \delta^g(y) = \sigma(z + \delta^f(z)).$$

From the definition of sigmoid activation function it easily follows that $0 \leq y + \delta^g(y) \leq 1$, which implies that $\delta^g(y)$ is a TBF. We now define the function $g(y) = \sigma(z_i + f(z)) - \sigma(z_i)$, where $i = \arg\max_i z_i$. It's easy to see that this g is the function associated to our δ^g. In fact:

$$\begin{aligned}\delta^g(y)_i &= \sigma(z + \delta^f(z))_i - \sigma(z)_i \\ &= \begin{cases} g(y) & \text{if } i = \arg\max_j y_j \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

Therefore, Theorem 3.1.2 guarantees that $\delta^g(y)$ is a minimal TBF under the Gödel t-conorm and the L_p norm.

$\square$

We note a few additional details: σ is monotonic increasing, which means that the highest preactivation corresponds to the highest activation:

$$\underset{j=1}{\operatorname{argmax}}{}_{j=1}^{n} \sigma\left(z_j\right) = \underset{j=1}{\operatorname{argmax}} z_j .$$

Another implication of the monotonicity of σ is that increasing a preactivation produces also an increase of the corresponding activation:

$$f(z) \geq 0 \Rightarrow \sigma\left(z_i + f(z)\right) \geq \sigma\left(z_i\right)$$

Putting these two observations together we can see that increasing the highest preactivation does indeed imply an increase of the highest activation. Also, note that $\delta^g(y)$ 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:

$$\delta_s^{w_c}(z)_i = w_c \cdot \text{softmax}(z)_i = w_c \cdot \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}. \qquad (3.11)$$

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

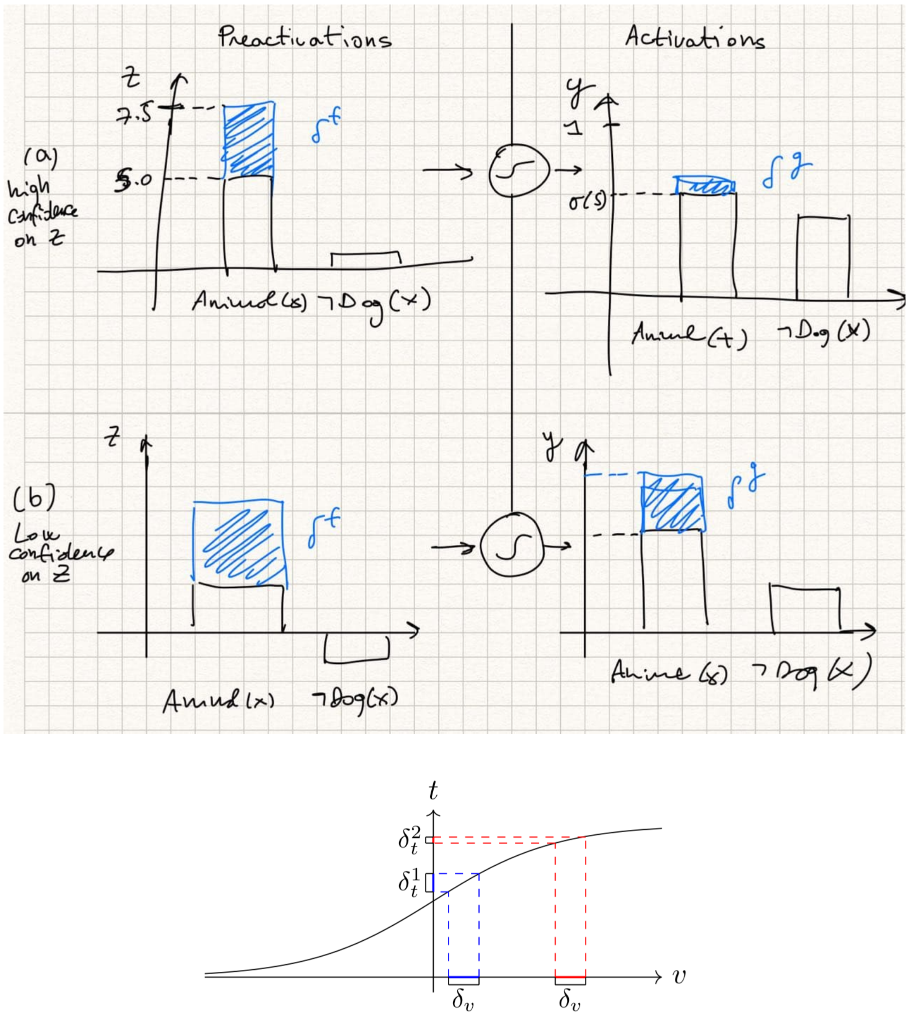

Figure 3.1: This image illustrates the actual deltas produced by KENN, which are the δ^f, opposed to the actual delta produced on the activations, which is δ^g. As we said, δ^g is not produced directly by the model but it is indirectly *induced* by the application of δ^f on the preactivations. This also 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.

A is a generic predicate of our language. We also define $z_A = \sigma^{-1}(y_A)$. Now, we note that equation (3.11), 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 $z_c = (z_{l_1}, \dots, z_{l_k})$ 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:

$$\delta^c_A = \begin{cases} \delta_s^{w_c}(z_c)_A & \text{if } A \in c \\ -\delta_s^{w_c}(z_c)_{\neg A} & \text{if } \neg A \in c \ , \quad \forall A \in \mathcal{K} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

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:

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

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. That would suffice in the cases where the knowledge is constituted only by a single clause, but in real applications

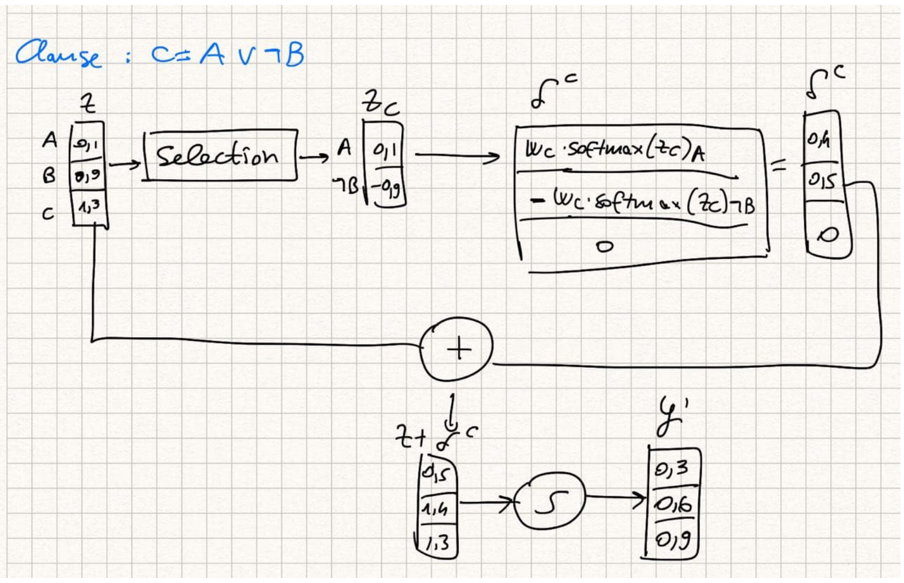

Figure 3.2: Summary of all the steps needed to produce δ^c, the vector of deltas derived from a single clause. We refer to this process as *clause enhancement*.

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:

$$y' = \sigma(z + \sum_{c \in \mathcal{K}} \delta^c). \tag{3.13}$$

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 deltas may cancel out rendering the contributions of the two clauses less effective.

3.2 KENN Architecture

In this section we present the architecture of the KENN layer in details. To be more precise, the architecture we are about to describe is valid only for unary predicates, meaning that only unary clauses will work as base knowledge. In the next section (ref TODO) we will also see that KENN is capable of dealing with binary clauses, but that will require a modification of the architecture.

As described above, the core functionality of KENN is the *clause enhancement*, i.e. the creation of a vector δ which, summed to the vector of predictions y, produces a modified vector of predictions y' which increases the truth value of the relative clause. The architecture that takes care of this task is the Clause Enhancer (CE): this submodule takes in input the full vector of preactivations from the original NN and computes the vector of deltas δ^c described in equation (3.12). For each clause in the base knowledge, a CE will be instantiated. The details of the architecture of the CE are described in Figure 3.3.

The CE performs the following operations:

1. takes the preactivations...
2. perform the select step..
3. creates delta according to eq...
4. transforms back from zof literals to z of predicates ...

Once the delta for each clause (δ^c) is produced by its corresponding CE, the next step is to aggregate all the deltas by summing them. The module that performs this operation is called the *Knowledge Enhancer* (KE), which is shown in Figure 3.4. More specifically, the KE performs the following operations:

1. Takes as inputs all the δ^c for each clause c in the knowledge and sums them.
2. It sums the obtained delta with the original preactivations
3. Applies the sigmoid activation function.

These steps produce the final vector of modified predictions y'. Notice that the KE is just the implementation of equation (3.13).

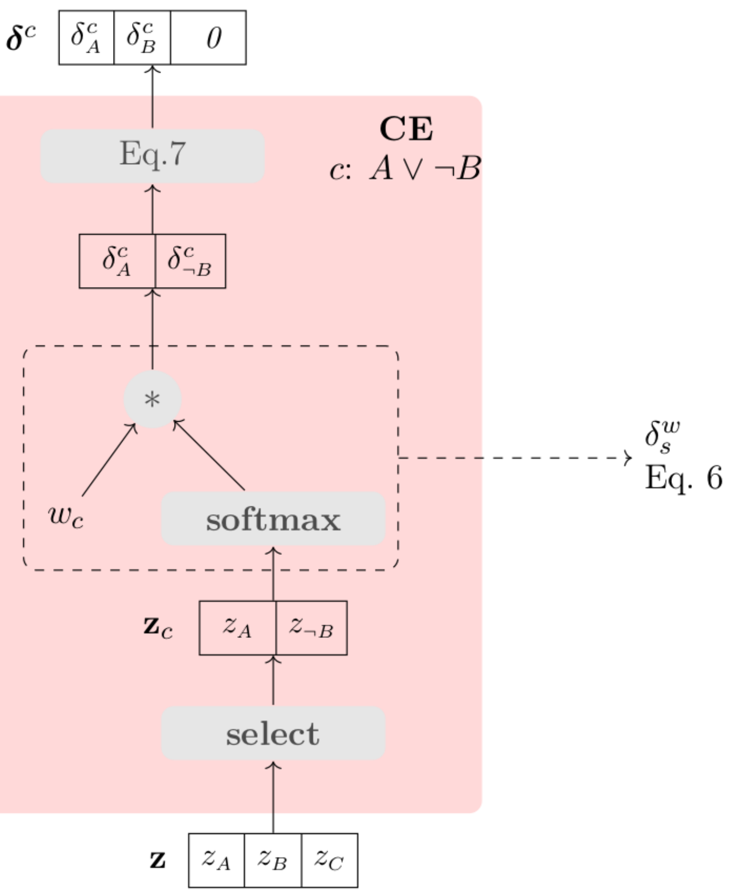

Figure 3.3: Detailed depiction of the Clause Enhancer for the clause.... todo

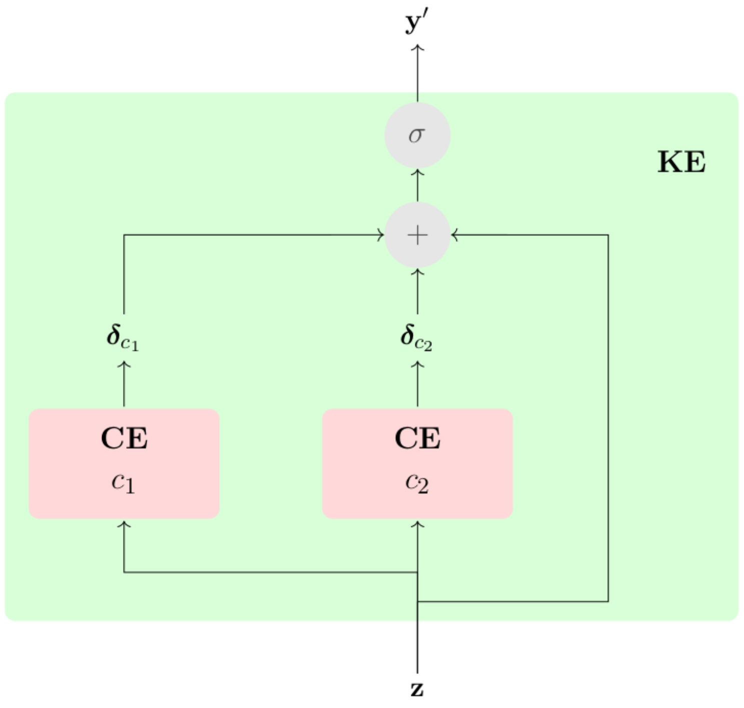

Figure 3.4: The KE architecture... todo

3.3 KENN for relational data

3.4 Experiments

3.5 Explainability in KENN

In che modo KENN è un modello explainable/interpretable. Prima fare qualche collegamento con quanto visto nel capitolo dell'explainability...

1. clause weights -> quanto le clausole influiscono sulle predizioni
2. ispezionare i delta permette di vedere direttamente come è stata modificata una predizione e perché;
3. possibilità di definire metriche (rank by improvement etc) che permettono di capire bene quali predizioni sono state migliorate, quali peggiorate, e a causa di quale clausola.
4. Interpretability: si può capire come KENN impari i clause weights e quanto è legato alla ”verità di una clausola”-> correlazione tra clause compliance e clause weights.

4 Conclusion

The conclusion goes here.

References

[1] L. Gilpin, D. Bau, B. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," Oct. 2018, pp. 80–89.

[2] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," 2015. [Online]. Available: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

[3] G. Marcus, "Deep learning: A critical appraisal," Jan. 2018.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 12 2013.

[5] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," 06 2015, pp. 427–436.

[6] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," 01 2017, pp. 2021–2031.

[7] T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and A. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 4356–4364.

[8] ACM. (2017) Statement on algorithmic transparency and accountability. [Online]. Available: https://www.acm.org/binaries/content/assets/public-policy/2017_usacm_statement_algorithms.pdf

[9] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (xai) program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, Jun. 2019. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2850

[10] Z. Lipton, "The mythos of model interpretability," *Communications of the ACM*, vol. 61, 10 2016.

[11] F. Doshi-Velez and B. Kim. (2017) Towards a rigorous science of interpretable machine learning. [Online]. Available: https://arxiv.org/abs/1702.08608

[12] B. Kim, "Interactive and interpretable machine learning models for human machine collaboration," 2015.

[13] M. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 02 2016, pp. 97–101.

[14] N. Burkart and M. Huber, "A survey on the explainability of supervised machine learning," *Journal of Artificial Intelligence Research*, vol. 70, 01 2021.

[15] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[16] B. Herman, "The promise and peril of human evaluation for model interpretability," *ArXiv*, vol. abs/1711.07414, 2017.

[17] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 0210–0215.

[18] S. Krening, B. Harrison, K. M. Feigh, C. L. Isbell, M. Riedl, and A. Thomaz, "Learning from explanations using sentiment and advice in rl," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, no. 1, pp. 44–55, 2017.

[19] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in *Proceedings of the 7th ACM Conference on Recommender Systems*, ser. RecSys '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 165–172. [Online]. Available: https://doi.org/10.1145/2507157.2507163

[20] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[21] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," 11 2014.

[22] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *preprint*, 12 2013.

Acknowledgments

This is the acknowledgments section.