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 Networks
TBF	t-conorm Boost Function
CE	Clause Enhancer
KE	Knowledge Enhancer
SBR	Semantic Based Regularization
LTN	Logic Tensor Networks
RNM	Relational Neural Machines
NSI	Neural Symbolic Integration

1 Introduction

STILL WIP

2

Explainability in Machine Learning

In recent years, research on Deep Learning (DL) has achieved remarkable results in a wide variety of domains, going from image recognition, speech recognition, machine translation, playing games, and many others, even outperforming human capabilities. Although DL is not a new research topic, its popularity and capabilities skyrocketed only in the last decade: this has been possible mostly thanks to the always growing availability of new data, together with the increase in computational power of modern machines. Despite the quick and huge success, researchers in the field have already shown some perplexities and moved some critiques towards this approach [2, 3]: is DL really the future of Artificial Intelligence (AI)? Will Neural Networks (NN) be able to give a good approximation of the human brain? Leaving aside such overwhelming questions, we should still be interested in what DL is capable to do at the moment, and what capabilities it still lacks. One particular aspect for which DL has been criticized is its lack of transparency: in fact, deep models 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, since no human supervisor can, ultimately, interpret what the model has learned by simply inspecting its internal structure. To refer to this undesirable property of deep NNs, the term "black box" is commonly used. DL models also present other critical and perplexing 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 [5]. Similarly, authors in [6] show how deep neural networks are easily fooled into misclassifying inputs with no resemblance to their true category. This kind of issues pose serious doubts about the ability of NN to learn general representations: indeed, if such networks can generalize well, how can they be confused by what we see as nearly indistinguishable images? Adversarial examples are not confined to the field of computer vision: natural language networks can also be fooled as shown in [7, 8]. Furthermore, it has been found that in several applications, DL models present strong biasedness. One example is reported in [9], 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 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" [10]. The need for XAI has been also expressed by the statement on algorithmic transparency and accountability released by the Association for Computing Machinery [11], and by the XAI program launched by DARPA in 2017 [12].

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. In this chapter, we try to provide a more rigorous definition for such terms, by reviewing what has been done in the literature so far.

2.1 Why do we need explainability

Before determining a good definition for *explainability* or *interpretability*, we must have a good understanding of what the real world objectives of XAI research are. More specifically, what are the desiderata of XAI which are still not being satisfied by the current ML tools and practices. 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 which is computed in order to answer in a quantitative way to some questions, like "how good is Model A able to generalize with respect to Model B?", or "what is the probability that Model A will misclassify an unseen sample?". This evaluation framework provides satisfactory answers for some kinds of questions, but still fails to answer other ones, especially those demanding things qualitative information like "why did Model A predict sample x to belong to class k?", or "how did Model A understand that sample x belongs to class k?". This kind of questions are the ones sought by XAI research. However, we argue that a rigorous definition for the desiderata of XAI cannot consist in a list of potential questions: this approach is qualitative and already tackled by philosiphical works [13]. We therefore need to express such needs in other forms other than simple questions.

Lipton [14] suggests that the need for explainability arises "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 [15] 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, which 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 could 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). However, this kind of properties are not easy to encode in ML systems and, at the same time, biases in the data can often lead to unethical decisions if not treated properly.

Some papers [16, 17] 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, at least 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 [18] provide a fairly exhaustive list of domains in need for explainability:

- **Medical Domain/Health-Care**: when the lives of humans are at stake, the need for explanations and knowledge are of paramount importance; take as an example a model used from researchers in order to associate to each patient a risk of suffering a certain disease. Such a model should not only be accurate, but intelligible from doctors: in this way they could understand the underlying causes for such disease, effectively advancing research in the medical domain;

- **Judicial System**: machine learning systems have also been explored for the automatic decision of judgement results [19]. Such systems should help judges and lawyers to take decisions, but in order to do so their decision must be well motivated;

- **Banking/Finance**: typical examples of automatic decision making in the banking domain are automatic credit approval systems. Since banks are legally obliged to provide customers with motivations when their credit request is denied, the usage of explainable models is required;

- **Automobile Industry**: autonomous driving systems are one of the most common and popular applications in DL research. Such autonomous agents are responsible for any accident that could take place on the road: for this reason explanations for each of the agent's decision must be provided, both for legal and security reasons, so that the system can be quickly fixed and improved;

- **Recommender Systems**: explainable recommendations boost the trustworthiness and effectiveness of recommender systems [20]. Furthermore, regulations like the aforementioned GDPR are currently requiring models that use users' personal data to provide predictions, to also provide explanations.

Examples of other domains requiring explainability include bio-informatics, marketing, election campaigns, precision agriculture or expert systems for the military.

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 [21], for example, the authors claim that explainability is a property of a model that implies interpretability, but not viceversa. More specifically they provide definitions that, we argue, are the most agreed upon in the literature. For clarity, from now on we will interpret those two terms with the following meanings:

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.

Explanations, according to the authors, can be evaluated in two ways: according to their intelligibility (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 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 also suggests that the explanation methods should be evaluated according to how such explanations behave on the curve from maximum interpretability to maximum completeness. This approach is followed in depth in [17], where the authors devise an explanation technique able to work for any classifier, by optimizing the intelligibility-completeness tradeoff. More information about this work is provided in Section 2.3.2.

In [15] interpretability is defined as the ability to explain or to present in understandable terms to a human. According to [18], 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 [22] 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.

Going back to our definitions, in general, it is true that an explainable model is almost always interpretable, but the viceversa might not always be true. A notable exception to this rule, however, is the human brain itself: 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 [23] warns its readers, making a clear distinction between descriptive and persuasive explanations: indeed implicit cognitive biases of the human brain could mislead us to trust wrong explanations (for example, humans naturally tend to prefer simpler descriptions). To avoid falling in this problem, one should always keep in mind the intelligibility-completeness tradeoff mentioned above.

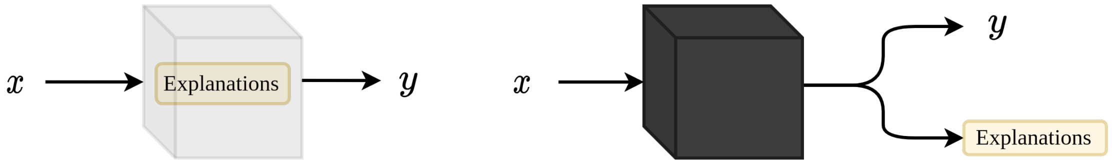


Figure 2.1: Intuitive representation of explanations coming from transparent models (left) vs. explanations coming from post-hoc interpretability techniques (right). Transparent models are inherently interpretable by their design; they can be inspected and explanations can be deduced from them. Post-hoc interpretability, on the other hand, refers to a wide range of techniques with which explanations are extracted by any already trained model.

It is clear that, even after providing some sort of definitions, the meanings of interpretability and explainability are still very generic and slippery. Nevertheless, the volume of research in XAI is quickly expanding, making the number of available methodologies continuously grow. One fundamental problem in XAI is the definition of specific properties, which make models explainable or interpretable. In the literature, two paradigms are often distinguished [24, 14]:

- **Transparency**, or integrated interpretability: it is mostly a feature of *interpretable* models. A transparent model can be interpreted by a human thanks to its own easy to understand design.

- **Post-hoc** interpretability: refers to that approach with which explanations are extracted from already trained models. Such models can be transparent, or retain their black box structure.

An intuitive illustration of those two concepts are illustrated in Figure 2.1.

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 [14] 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. This notion of transparency is not very applicable to modern machine learning techniques and is often disregarded.

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. Transparent models are fascinating, but recent research in DL has proven that predictive performances rise when building deeper models, and not vice-versa. For this reason, we argue that the most promising techniques in XAI come from the research of post-hoc explainability methods.

2.3.2 Post-hoc interpretations

With post-hoc interpretability we refer 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 on the performances of the model, which is treated as a black box. Unlike transparency, this kind of interpretability is the one that applies to humans. Lipton [14] summarizes post-hoc explainability methods into four main categories: text explanations, visualizations, local explanations and explanations by example.

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 [25], in recommender systems: [26]

(Non ho ancora trovato molti esempi di metodi di questo tipo, forse la tolgo come categoria)

Visualization

Another popular approach for 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 [27], 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 [1], 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).

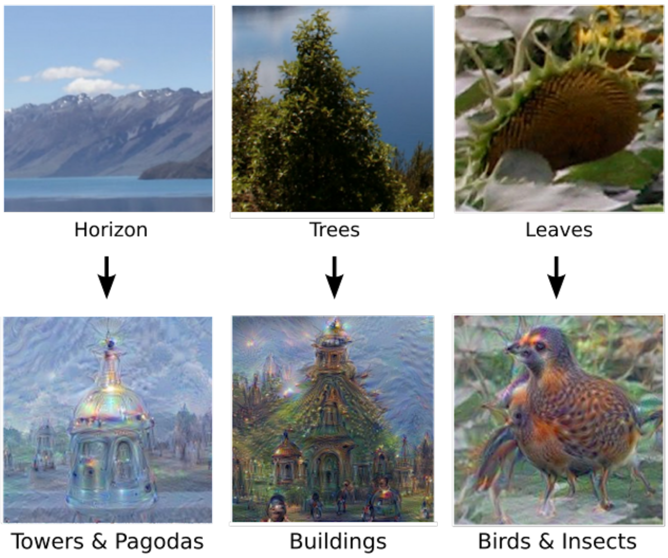


Figure 2.2: Process of image manipulation shown in [1]. From an intuitive point of view, starting from existing real images, the network is asked to enhance the features of the image that resemble the desired object (in this case, starting from an image of a tree, start looking for features that resemble 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.

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 [28], 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

With visualization techniques, one aims to understand what the model learned from a global point of view: for example, the activation maximization technique described earlier will reflect the "internal representation" that the model has of a particular class. With local explanations, instead, one aims at giving explanations in the context of a specific example. One popular approach is the computation of saliency maps, or relevance scores [29, 22, 30]. Specifically, given a data point $x \in \mathbb{R}^k$, and given the learned function f, the predicted class for point x will be $f(x)$; this approach aims to assign to each feature a score $R(x)_i$, representing a measure of how relevant the feature x_i is for explaining $f(x)$. One simple approach to obtain the relevance scores is *sensitivity analisys*: it consists on finding the input features for which the output is more sensitive, meaning the ones that best contribute to increase the value of the output. In mathematical terms, it is defined as:

$$R(x)_i = \left(\frac{\partial f}{\partial x_i}\right)^2,$$

where the gradient is then evaluated on x. Such relevance scores can then be visualized: for example, in images they can be represented as a mask. We should note a subtle detail: this method gives us an explanation of the *variation* of $f(x)$, not of the value of $f(x)$ itself. In other terms, this method does not answer the question *"what parts of x make it belong to class y?"*, but instead it answers *"what parts of x make it belong more/less to class y?"*.

While the relevance scores can be seen as a way of extracting explanations, those can also be used in different ways. An interesting approach can be found in [31], where the authors explicitly train the relevance scores in a supervised manner, in such a way to make them conform to some manually curated notion of where the attention should be put. Following this approach, the authors aim to train a model which is "right for the right reason". The motivation is that, if the assumption that relevance score faithfully describes the model's underlying behavior, then costraining such relevance scores in order to match domain knowledge would result in a model that, in some way, will use that domain knowledge to take decisions.

Another interesting approach for generating local explanations is from Ribeiro et al. [17]: they propose an explanation technique called LIME (Local Interpretable Model-agnostic Explanations), which purpose is to locally approximate complex models with simpler, interpretable models, following the completeness-interpretability tradeoff mentioned by Gilpin in [21] . More specifically, they define an explanation as an interpretable model $g \in G$, where G is a set of models which, with the right conditions, can be considered interpretable (e.g. decision trees). In fact, the authors note that even those that are considered transparent models, will not be considered interpretable by a human supervisor if the model relies on thousands of features to make the prediction. This means that there should be a measure of complexity that determines how much a "potentially interpretable" model g is actually interpretable. They define such measure of complexity as $\Omega(g)$ (e.g. for decision trees, this could represent the depth of the tree). Then,they denote with $f : \mathbb{R}^k \to \mathbb{R}$ the model to be approximated, and with $\pi_x(z)$ a proximity measure between a feature vector x and z. Finally, $\mathcal{L}(f, g, \pi_x)$ is defined as a measure of how badly the model g approximates f near the input sample x. With this setup, one can find a model g which is maximally interpretable, and which best explains the original model, near a specific input. Specifically, this model is the output of LIME, which is defined as follow:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}\left(f, g, \pi_x\right) + \Omega(g).$$

Another recent technique which provides local explanations is the attention mechanism [32], which, in recent years, has proven to be very successful mostly in machine translation [33] and computer vision [34]. Despite not being explicitly designed to provide explanations, they can do so by visualizing their internal attention scores, which highlight sections of the input data which were most influential for the classification. It is interesting to note that this kind of explanation is almost identical to the one provided by saliency maps, discussed above. The core difference is that, for the attention mechanism, the attention scores are directly computed by the model at inference time, while the saliency maps are obtained after inference via backpropagation. It is interesting to know that datasets depicting how humans distribute attention have been created [35, 36]: this can allow us to evaluate attention-based models according to how much their attention patterns conform with the human ones.

Explanation by example

A common way in which humans justify their decisions is by offering analogies between the current object in study, and similar objects. For example, to decide the best treatment for a medical condition, doctors often refer to previous similar case studies. This kind of explanation is referred to explanation by example. The first approach with this method was proposed in [37]: the authors aim to generate explanations in the form of a collection of elements from the training set, returned by the model (in this case, a neural network) together with the actual prediction. To do this, they generate a distance metric directly from the model: in this way it is possible, at inference time, to scan the entire training set and look for the data points which, by the model's point of view, are the most similar to the current one being classified. Given x the current sample being predicted and given y any element of the training set, this metric is defined as the euclidean distance between the hidden representations of y, and x.

2.4 Neural Symbolic Integration

Despite the attempts in the literature to give rigorous definitions of explainability and interpretability, those remain slippery concepts: interpretations may vary depending on the application domain and type of task to be performed. The need for explainability is mostly due to the black box nature of ML models, which is universally agreed upon. Indeed, differently from humans, ML models employ only sub-symbolic representations of concepts, meaning that everything inside the model is encoded as tensors of real values, which are inherently opaque to humans. Furthermore, classic ML systems learn everything from data, which is a radically different paradigm from the one humans use for learning. This has been identified as one of their major flaws [2]: NNs can offer outstanding performances when the problem is confined inside a specific domain (e.g. recognize spoken words in a short audio clip), but have nothing to offer when it comes to trivial tasks like commonsense reasoning. One possible way to extend the range of applications of NNs, could be to equip them with the ability to reason with abstract symbolic terms. Indeed, the introduction of symbolic language inside neural networks could also help in the process of encoding general knowledge inside them, a problem for which the formulation is still considered incomplete (cfr. Section 2.1).

The area of research that tackles the problem of integrating symbolical knowledge with neural architectures is known as Neural Symbolic Integration. The intuition that motivates NSI as field of research goes beyond explainability: while neural networks give good evidence to be a good modeling framework for the human mind (e.g. parallelization and adaptive learning from the environment) they completely lack the ability to reason in symbolic terms [38]. Indeed, works in NSI argue that logic is the best tool to represent general knowledge, and their first aim is to build systems capable of dealing with both symbolic and sub-symbolic representations. This is a well known, crucial task, and not an easy one: the problem of integrating the statistical nature of learning with the logical nature of reasoning has been already identified as one of the most important research challenges in computer science [39].

In the next chapter, we delve in the details of Knowledge Enhanced Neural Networks (KENN) [40], a neural network layer designed to integrate logical knowledge in NNs. We will also provide a short summary of other notable methods in NSI and provide a comparison of experimental results.

3

Knowledge Enhanced Neural Networks

Knowledge Enhanced Neural Networks (KENN) [40] 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)**.** The Prior Knowledge is defined as a 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 $\mathcal{P} = \{p_1, \dots, p_q\}$. A predicate is a symbol, which can take in input a number of constants $k \geq 1$. If $k = 1$, the predicate represents a property of the input constant; if $k > 1$, it represents a relation between the input constants. k is called the *arity* of the predicate.

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

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

In this equation, l_i is a literal, i.e. a formula constituted only by a n-ary predicate, or its negation. Specifically, when writing $l_i \neq l_j$, we mean that those two literals never share their constituting predicate. For example, the clause $A(x) \vee \neg B(x)$ is considered an acceptable clause, while $A(x) \vee B(x) \vee \neg A(x)$ is not. Also clauses have an arity, which is by definition the number of variables given in input to the clause.

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, in classical logic, 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 $c[x_1/k_1, \dots, x_n/k_n]$, is the clause obtained by substituting k_i to $x_i, \forall i = 1, \dots, n$.

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 concatenation of 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.

With those definitions, we can already see how this theoretical framework is suitable to describe a NN. 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, for classification tasks, are always expressed in terms of values in the interval $[0, 1]$. The natural consequence of this choice is that, from this point on, we will have to rely on the rules of Fuzzy Logic [41], 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 $\perp$: $[0, 1]^2 \to [0, 1]$ 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:

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

*Writing $1 - \mathcal{I}(P)$ is a slight abuse of notation since $\mathcal{I}(P)$ is a function. To be more precise one shoud write $\mathcal{I}(\neg P)(\mathcal{I}(c)) = 1 - \mathcal{I}(P)(\mathcal{I}(c)), \forall c$.

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:

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

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 a collection of feature vectors $\{x_1, \ldots, x_d\}$ (where d is the arity of c), then the truth value of the grounded clause predicted by the NN will be $\perp((y_c)_1, \ldots, (y_c)_k)$, where:

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

The intuition behind KENN is very simple: given the vector y of predictions of 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 any t-conorm $\perp$ and $\delta \in \Delta$, it holds that:

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

Proof. By definition of TBF, $\delta(t)_i \geq 0$. 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 $\perp(t) \leq \perp(t + \delta(t))$. □

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 the 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 literal, 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 $\|\delta(t)\|_p < \|\delta^f(t)\|_p$. We have to show that

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

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 $\delta^f(t)$.

□

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 σ takes only values in $[0, 1]$ 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 $0 \leq y + \delta^g(y) \leq 1$. To prove that δ^g is a TBF we have to show that $\delta^g(y) \geq 0$. We note a few details: σ is monotonic increasing, which means that:

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

Another implication of the monotonicity of σ is that:

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

This implies that $\delta^g(y) = \sigma(z + \delta^f(z)) - \sigma(z) \geq 0$. 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.

□

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, $\lim_{|z| \to \infty} |\sigma(z + \delta^f(z)) - \sigma(z)| = 0$. 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:

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

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 A is a generic grounded 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

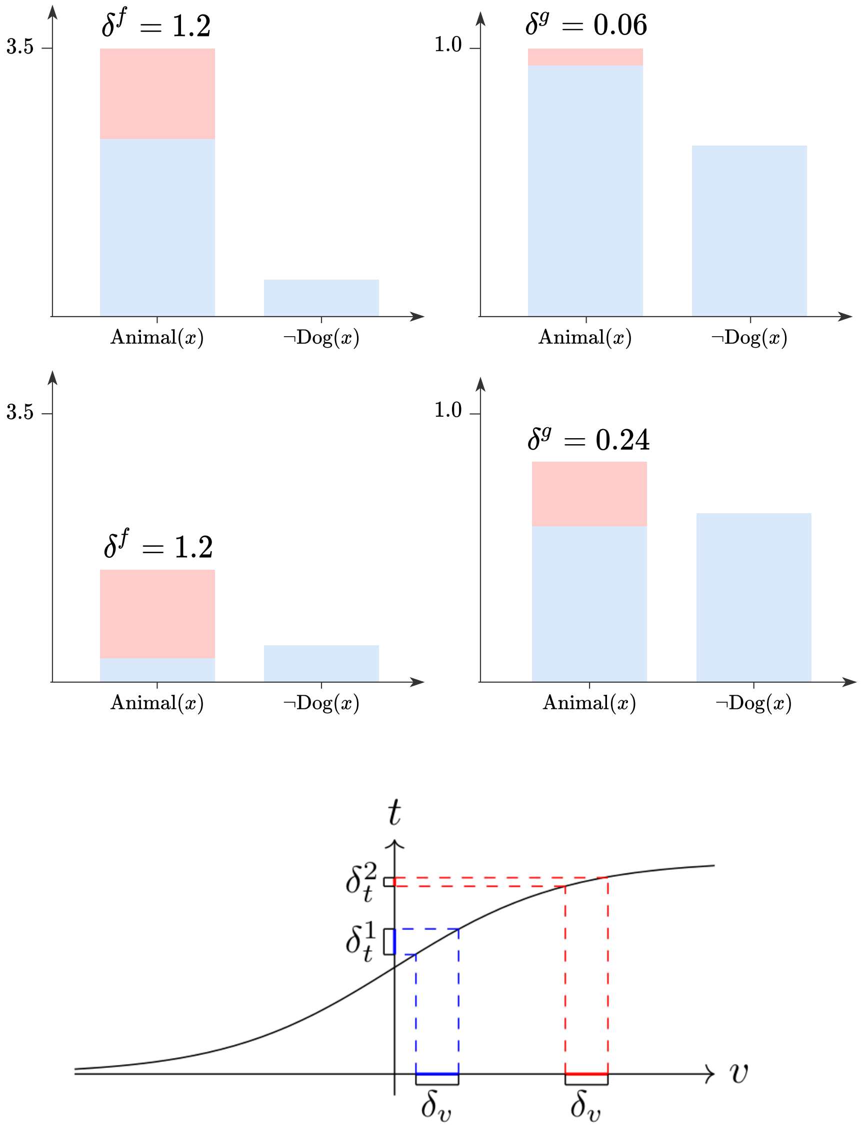


Figure 3.1: This image illustrates the actual deltas produced by KENN, which are the δ^f, opposed to the actual delta produced on the activations, which is δ^g. As we said, δ^g is not produced directly by the model but it is indirectly *induced* by the application of δ^f on the preactivations. This also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

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:

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

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:

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

In Figure 3.2, an example of computation of the δ^c vector is provided.

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

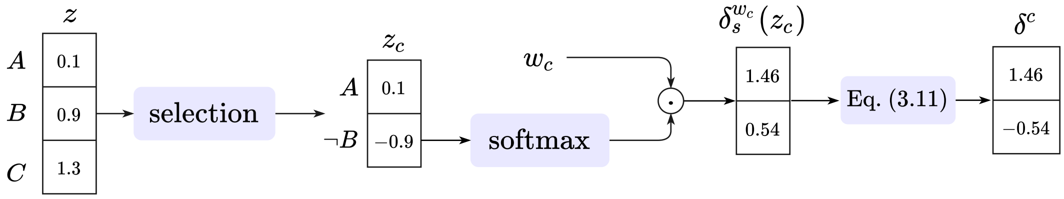


Figure 3.2: Example with all the steps needed to compute δ^c for the clause $A \vee \neg B$ starting from the vector of preactivations z; for this example $w_c = 2$. We refer to this process as *clause enhancement*.

the cases where the knowledge is constituted only by a single clause, 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:

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

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 general, KENN is capable of dealing also with binary clauses, but that will require a modification of the architecture, as we'll see later.

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 3.3a.

The CE performs the following operations:

1. Takes in input the vector of preactivations z coming from the base NN;
2. Performs the *selection* step;
3. Computes the vector of deltas according to equation (3.11);
4. Transforms the delta vector according to equation (3.12)

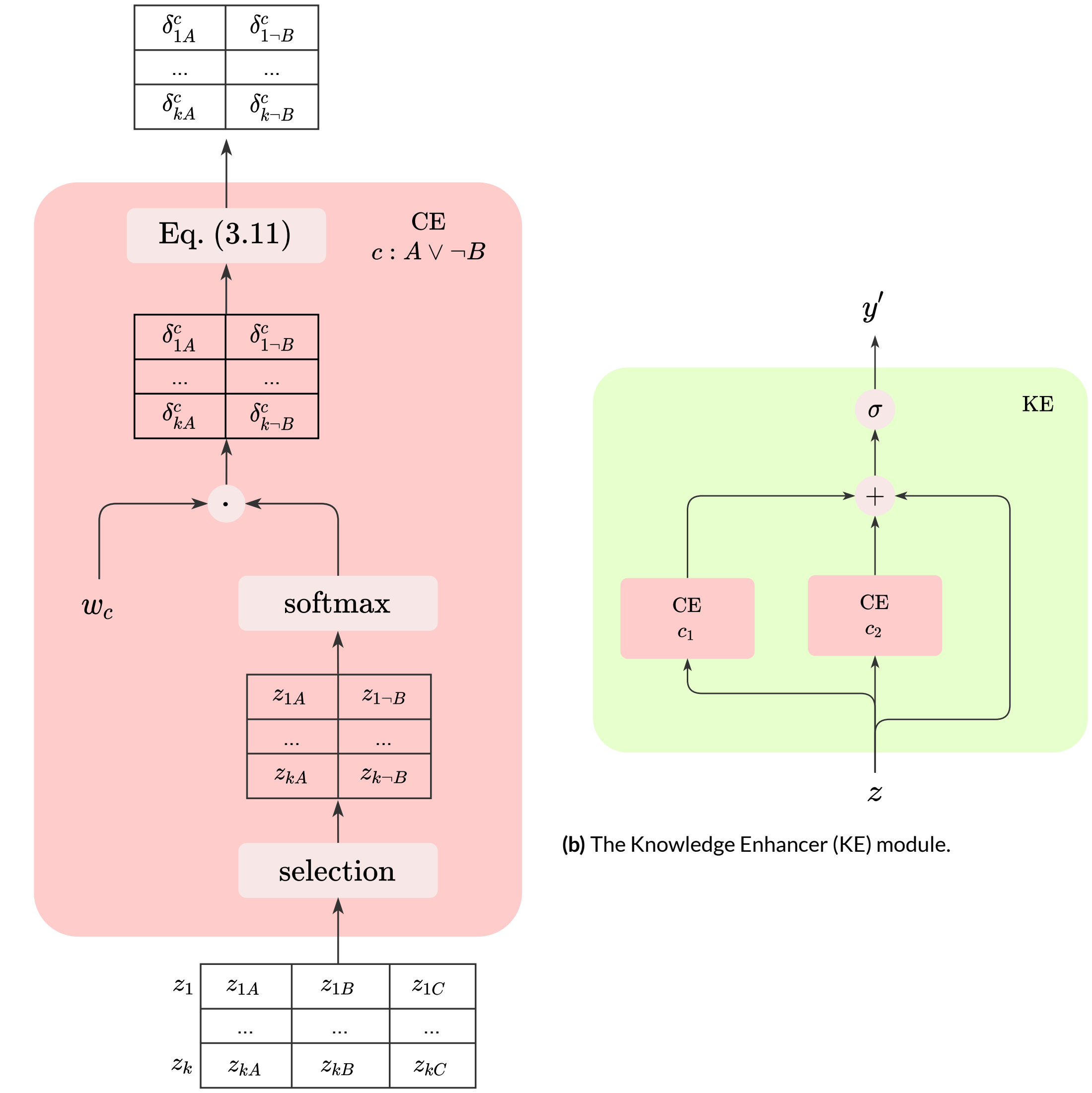
Note that, in practice, the CE does not take in input a single vector of preactivations at a time, but works in parallel taking in input mini-batches of data. Once every CE has produced a vector of deltas, the next step is to aggregate them by summing them up. The module that performs this operation is called the *Knowledge Enhancer* (KE), which is shown in Figure 3.3b. More specifically, the KE performs the following operations:

1. Takes as inputs all the δ^c for each clause c in the Base Knowledge and sums them;
2. It sums the obtained delta with the original preactivations vector;
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).

3.3 KENN for relational data

As we said, KENN can also handle relational data; specifically, thanks to its general approach, KENN can handle clauses with any arity. Nevertheless, for simplicity, we will just describe the binary case. In order to integrate binary clauses, the first step is to split the base knowledge into two complementary subsets of clauses $\mathcal{K}_U$ and $\mathcal{K}_B$, containing the unary and binary clauses respectively. In this way, the set of all the clauses constituting the Prior Knowledge is denoted as $\mathcal{K} = \mathcal{K}_U \cup \mathcal{K}_B$. The intuition at the base of KENN remains the same: starting from initial predictions z, we will need to produce a vector of deltas δ. The only difference, is that δ will be the sum of two different deltas, one produced just by unary clauses and the other just by binary clauses. More simply, this consists in modifying equation (3.13) as follows:


δ^c_{1A} $\delta^c_{1\neg B}$
... ...
δ^c_{kA} $\delta^c_{k\neg B}$
Eq. (3.11)
CE
$c: A \vee \neg B$
δ^c_{1A} $\delta^c_{1\neg B}$
... ...
δ^c_{kA} $\delta^c_{k\neg B}$
$\cdot$
w_c
softmax
z_{1A} $z_{1\neg B}$
... ...
z_{kA} $z_{k\neg B}$
selection
z_1 z_{1A} z_{1B} z_{1C}
... ...
z_k z_{kA} z_{kB} z_{kC}
y'
σ
KE
$+$
CE
c_1
CE
c_2
z

(b) The Knowledge Enhancer (KE) module.

(a) The Clause Enhancer (CE) module.

Figure 3.3: Illustration of the KENN architecture. Images are replications of the illustrations provided in [40].

$$y' = \sigma(z + \sum_{c \in \mathcal{K}_U} \delta^c + \sum_{c \in \mathcal{K}_B} \delta^c). \tag{3.14}$$

Note that the procedure to compute the quantity $\sum_{c \in \mathcal{K}_U} \delta^c$ is the one we described before, since all is limited to unary clauses. The only thing left to define is how to compute a delta vector starting from a binary clause.

Table representation of binary data

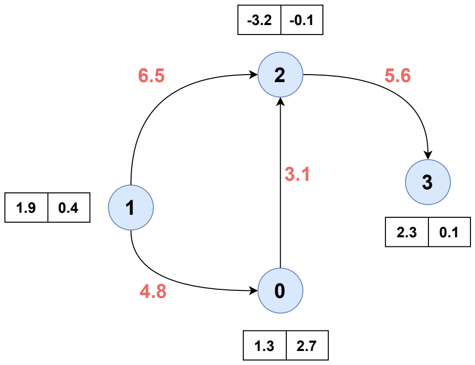
In order to deal with binary clauses, KENN represents truth values of every grounded predicate inside two different matrices: U and B. Matrix U has one row for each constant object, one column for each unary predicate and an additional column containing an identifier for each object. Matrix B, instead, has one row for each couple of objects: specifically, for each pair of objects such that there is a binary relation between them. The first two columns of B contain the identifier indices of the pair of objects, while the remaining columns contain the preactivations for each binary predicate, grounded on that pair of objects. So, in the case of n unary predicates, m binary predicates, p total objects and q couples of objects, U and B will have dimensions $p \times (n+1)$ and $q \times (m+2)$ respectively. Figure 3.4 shows an example with a visualization of U and B. With those newly defined matrices, equation (3.14) can be implemented as follows:

$$\begin{cases} y_u = \sigma(U + \delta U) \\ y_b = \sigma(B + \delta B) \end{cases} \tag{3.15}$$

where y_u and y_b are the final predictions for the unary and binary predicates respectively, and δU and δB are the delta matrices that need to be computed.

The JOIN operation

At this stage, we have two matrices, U and B, containing preactivations of unary and binary predicates respectively. However, recall that the CE module takes in input a single matrix containing all the preactivations, in order to compute the deltas. The fact is that, given any binary clause, we must be able to access the preactivation value of any of its atoms, from a single matrix. Take as an example the grounded clause $A(c_1) \vee B(c_1, c_2) \vee \neg A(c_2)$. The preactivations of $A(c_1)$ and $A(c_2)$ can be accessed from matrix U, but the one for $B(c_1, c_2)$ can be accessed only from B. For this reason, we need a way to merge both the binary and unary preactivations into



U

i	Unary	
0	1.3	2.7
1	1.9	0.4
2	-3.2	-0.1
3	2.3	0.1

B

sx	sy	Binary
0	2	3.1
1	0	4.8
1	2	6.5
2	3	5.6

Figure 3.4: Representation of relational data inside KENN. Specifically, objects and relations can be seen as nodes and edges of a directed graph. The preactivations of each grounded predicate are represented in tables U and B. In this example, two unary predicates and one binary predicate are present. Note that in matrix B only the object pairs such that there is a relation between them are reported.

a single matrix. KENN does this with the JOIN operation.

Definition 3.3.1 (JOIN)**.** Given n unary predicates, m binary predicates and their respective matrices U and B, the JOIN operation is defined as follows:

$$\text{JOIN}(U, B) = M \tag{3.16}$$

where the i-th row of the matrix M is:

$$\begin{aligned} &M_i = (sx, sy, U_{sx,2}, \dots, U_{sx,n+1}, U_{sy,2}, \dots, U_{sy,n+1}, B_{i3}, \dots, B_{i(m+2)}), \\ &\text{where } sx = B_{i1}, sy = B_{i2}. \end{aligned} \tag{3.17}$$

More simply, M has the same number of rows as matrix B and shares its two first columns; additionally, it also contains the preactivations of all the unary predicates for both the first and second index of each row, together with all the preactivations for the binary predicates.

From this new matrix, the CE can access the preactivations for any atom of any grounded binary clause and can compute a delta matrix, which we will call δM.

The GROUP BY and SELECT operations

As we saw with our previous example, binary clauses can also contain unary predicates. This implies that the delta vectors generated from binary clauses can actually modify the preactivations of both unary and binary predicates. More specifically, this means that the matrix δM contains some information that is supposed to go both inside matrices δU and δB. KENN extracts the unary deltas from δM with the GROUP BY operation, and does the same for the binary deltas with the SELECT operation.

Definition 3.3.2 (GROUP BY)**.** The GROUP BY (GB) operation takes in input the matrix of deltas δM and an index (x or y), and is defined as follows:

$$\begin{cases} \text{GB}(\delta M, x) = \delta U_x \\ \text{GB}(\delta M, y) = \delta U_y \end{cases}$$

where

$$\begin{cases} (\delta U_x)_i = \left(i, \sum_{j \in J_x} \delta M_{j3}, \ldots, \sum_{j \in J_x} \delta M_{j(2+n)}\right) \\ (\delta U_y)_i = \left(i, \sum_{j \in J_y} \delta M_{j(n+3)}, \ldots, \sum_{j \in J_y} \delta M_{j(2n+2)}\right) \\ J_x := \{j : \delta M_{j1} = i\} \\ J_y := \{j : \delta M_{j2} = i\} \end{cases} \tag{3.18}$$

In simple terms, the GROUP BY operation extracts the deltas for the unary predicates from the delta matrix δM, by aggregating the results for each object identifier. This is exactly equivalent to a GROUP BY query in relational databases.

Definition 3.3.3 (SELECT)**.** The SELECT operation simply extracts the preactivations of the binary atoms from the δM matrix. Specifically:

$$\text{SELECT}(\delta M) = \delta B$$

where:

$$\delta B_i = \left(\delta M_{i1}, \delta M_{i2}, \delta M_{i(2n+3)}, \ldots, \delta M_{i(2n+m+2)}\right).$$

Putting it all together

Now that all the necessary tools have been defined, we are ready to describe the process to compute the matrices δU and δB. KENN uses two different KE to generate delta matrices: one

(KE$_u$) takes in input the matrix U, and returns a delta matrix δU_u, using the standard procedure for unary predicates described in the previous section. The other (KE$_b$) takes in input a matrix $M = \text{JOIN}(U, B)$, and returns a delta matrix δM. Afterwards, the following operations are performed:

1. The final delta matrix for binary predicates is obtained as follows: $\delta B = \text{SELECT}(\delta M)$;
2. Two matrices $\delta U_x = \text{GB}(\delta M, x)$ and $\delta U_y = \text{GB}(\delta M, y)$ are computed;
3. δU_x and δU_y are summed together, obtaining matrix δU_b;
4. The final delta matrix for unary predicates is obtained as follows: $\delta U = \delta U_u + \delta U_b$.

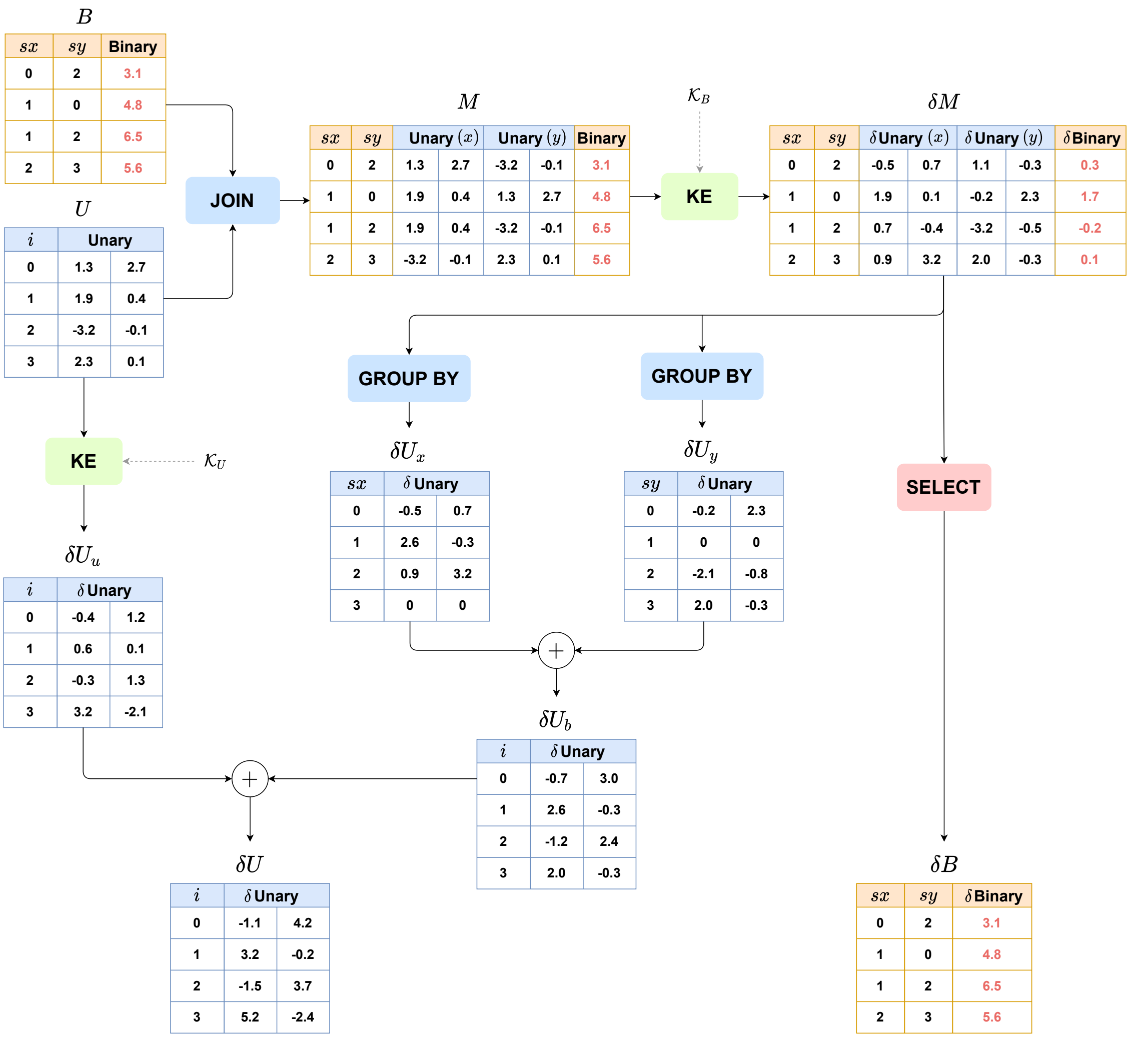
In Figure 3.5, we report an example with the whole process.

3.4 Related Work

As already introduced at the end of Chapter 2, the integration of statistical learning with logical symbolic knowledge is one of the key challenges in the research on artificial intelligence, specifically for the field of NSI. Authors of [40, 42] subdivide NSI sytstems into three main groups, based on their different objectives:

1. **Differentiable Reasoning** : research on this subfield aims at developing models capable of logical reasoning, meaning capable to derive logical knowledge, given a base of ground truth logical facts (axioms). (non sono sicurissimo sia questa la definizione...)
2. **Inductive Logic Programming** : here the goal is to extract logical knowledge from data, or to refine an existing one;
3. **Knowledge Guided Learning**: here the goal is learning in classical ML sense, where the knowledge has the role of a supervisor, meaning that the machine should learn according to the base knowledge.

We are interested in Knowledge Guided Learning (KGL), the class of models where KENN belongs. The objective is to improve the performance of a NN model, by providing a Prior Knowledge which is expressed in terms of logical formulas and which acts as a supervisor for the learning process. At present, two main ways of injecting knowledge inside NN have been employed. The first one involves the usage of a regularization term in the loss function. Indeed, logical rules can be interpreted as constraints for the weights in the learning process; in ML, the



B

sx	sy	Binary
0	2	3.1
1	0	4.8
1	2	6.5
2	3	5.6

U

i	Unary	
0	1.3	2.7
1	1.9	0.4
2	-3.2	-0.1
3	2.3	0.1

M

sx	sy	Unary (x)		Unary (y)		Binary
0	2	1.3	2.7	-3.2	-0.1	3.1
1	0	1.9	0.4	1.3	2.7	4.8
1	2	1.9	0.4	-3.2	-0.1	6.5
2	3	-3.2	-0.1	2.3	0.1	5.6

δM

sx	sy	δ Unary (x)		δ Unary (y)		δ Binary
0	2	-0.5	0.7	1.1	-0.3	0.3
1	0	1.9	0.1	-0.2	2.3	1.7
1	2	0.7	-0.4	-3.2	-0.5	-0.2
2	3	0.9	3.2	2.0	-0.3	0.1

δU_x

sx	δ Unary	
0	-0.5	0.7
1	2.6	-0.3
2	0.9	3.2
3	0	0

δU_y

sy	δ Unary	
0	-0.2	2.3
1	0	0
2	-2.1	-0.8
3	2.0	-0.3

δU_u

i	δ Unary	
0	-0.4	1.2
1	0.6	0.1
2	-0.3	1.3
3	3.2	-2.1

δU_b

i	δ Unary	
0	-0.7	3.0
1	2.6	-0.3
2	-1.2	2.4
3	2.0	-0.3

δU

i	δ Unary	
0	-1.1	4.2
1	3.2	-0.2
2	-1.5	3.7
3	5.2	-2.4

δB

sx	sy	δ Binary
0	2	3.1
1	0	4.8
1	2	6.5
2	3	5.6

Figure 3.5: This figure shows an example with all the necessary computations to compute the final delta matrices δU and δB (in the bottom), starting from matrices U and B (top left).

natural way to introduce constraints in learning is to add a penalization term in the loss function, which represents in some way the satisfaction of the logical rules. The second approach, adopted by KENN, is to directly modify the network architecture: in this way, logical rules are enforced at the level of the network topology.

3.4.1 Regularization Approaches

In KGL, regularization approaches enforce the satisfaction of the knowledge by defining a special penalization term to be applied to the loss function. Here we briefly mention two important examples of regularization based approaches: Logic Tensor Networks and Semantic Based Regularization.

Logic Tensor Networks

Logic Tensor Networks (LTN) [42] is a notable example of a method capable of integrating logical knowledge inside a NN by directly modifying the loss function. To do this, the authors define a differentiable first-order logic language called Real Logic, with which they manage to represent common deep learning tasks, such as clustering, multi-label classification, relational learning, regression, embedding learning and many others. In simple terms, the role of Real Logic is to act as a bridge between the purely symbolic world of logic, and the sub-symbolic world of neural systems. We give a brief summary of the core definitions in Real Logic, in order to understand how learning is possible.

Real Logic is defined over a first order language $\mathcal{L}$ with a signature containing a set of constant symbols (or objects) $\mathcal{C}$, a set of variable symbols $\mathcal{X}$, a set of functional symbols $\mathcal{F}$ and a set of relational symbols (or predicates) $\mathcal{P}$. We refer to the set $\mathcal{S} = \mathcal{C} \cup \mathcal{X} \cup \mathcal{F} \cup \mathcal{P}$ as the set of symbols of $\mathcal{L}$. Each symbol of the language can belong to different domains (i.e. can be of different types): for example the constant $c_1 \in \mathcal{C}$ can represent a specific person, while the constant $c_2 \in \mathcal{C}$ can represent a specific city. The same concept applies to functions and predicates. To represent the domain of each symbol, it is assumed that there exists a non-empty set $\mathcal{D}$, containing all the domains, which are in turn symbols. Then, to properly assign each symbol to its corresponding domain, the functions $\mathbf{D}$, $\mathbf{D_{in}}$ and $\mathbf{D_{out}}$ are defined:

$$\mathbf{D} : \mathcal{X} \cup \mathcal{C} \to \mathcal{D} \qquad \mathbf{D_{in}} : \mathcal{F} \cup \mathcal{P} \to \mathcal{D}^* \qquad \mathbf{D_{out}} : \mathcal{F} \to \mathcal{D},$$

where $\mathcal{D}^*$ is the set of all finite sequences of symbols in $\mathcal{D}$. The point where the symbolic language meets the subsymbolic world of neural networks is in the definition of grounding, which is the process in which each symbol is given its numeric representation. Indeed, in Real Logic, each domain is interpreted as sets of tensors in the real field. In the same way, each constant, variable and term of the language is interpreted as a tensor of real values, and function symbols are interpreted as functions between tensors. Predicates, are interpreted as functions that map tensors into the interval $[0, 1]$. So, given s any symbol of $\mathcal{L}$, its grounding is denoted as $\mathcal{G}(s)$. The authors then define how to compute the grounding of formulas, by using the semantics of first-order fuzzy logic. The way in which learning becomes possible is by defining parametric definition of grounding for symbols: given a symbol s, the parametric grounding of s is a grounding which is not known in advance, and can be computed exclusively by knowing a set of parameters. It is denoted as $\mathcal{G}(s|\theta_s)$, where θ_s is the set of parameter values that uniquely determines the value of the grounding. It's interesting to note that, based on what kind of symbol we are learning the grounding for, one can identify a corresponding task in machine learning:

- If $s \in \mathcal{C}$, corresponds to learning an embedding;
- If $s \in \mathcal{F}$, it corresponds to learning regression tasks;
- If $s \in \mathcal{P}$, it corresponds to learning a classification task.

For the simple case of binary classification, for example, the parametric grounding can be defined as $\mathcal{G}(A|\theta) : x \to \sigma(\text{MLP}_\theta(x))$

Semantic Based Regularization

Semantic Based Regularization (SBR) is a general learning framework designed to integrate domain specific background knowledge in the form of first-order logic (FOL) clauses. To enforce the satisfaction of all the clauses, SBR introduces special regularization terms in the loss function, which represent the satisfaction of the knowledge. Specifically, given a background knowledge represented by set of H clauses, the satisfaction of the h-th clause can be represented by the quantity denoted by $0 \leq \phi_h(f) \leq 1$, where f is the vector function learned by the model. From here, the regularization term to be added to the loss function is defined as

$$\sum_{h=1}^{H} \lambda_h (1 - \phi_h(f)),$$

where λ_h is the weight associated to the h-th constraint. A higher value of λ_h will increase the cost of not satisfying the constraint, meaning that the importance of the corresponding rule will increase. The conversion of FOL clauses into differentiable functions is made possible by considering fuzzy generalizations of FOL logic, similarly to how it is done in KENN. This is a natural approach for machine learning tasks, but a major disadvantage over KENN is that, since the clause weights are introduced at the level of the loss function, those cannot be learned and are required to be known in advance. This, of course, is unlikely to happen in real scenarios and it is much more desirable to learn the weights together with the other learnable parameters of the model.

3.4.2 Model Based Approaches

An example of model based approach for the integration of logic in NNs is represented by Relational Neural Machines (RNM) [44]. An RNM models a probability distribution over a set of m output variables, given the predictions provided by one (or more) base NN, and a set of model parameters. Specifically, differently from a standard NN, each forward pass in RNM is performed in two separate stages: in the first one, the predictions from the NN are obtained from the input data; the second is a *semantic* stage, where the logical costraints are enforced over the output. More precisely, this second stage is performed by an undirected graphical model. RNM defines a conditional probability distribution of the exponential family over the output variables, which is defined as follows:

$$p(y|f, \lambda) = \frac{1}{Z} \exp\left(\sum_{x \in S} \Phi_0(f(x), y(x)) + \sum_{c} \lambda_c \Phi_c(y) \right),$$

where Z is the partition function, f is the prediction from the NN, λ is the set of parameters driving the semantic stage, Φ_0 is a potential function which enforces the supervised learning paradigm, S is the subset of supervised input vectors and Φ_c is a potential associated to c, a FOL formula part of the base knowledge. Specifically, Φ_c enforces the satisfaction of the clause c, while λ_c determines the weight of such clause.

The approach used by RNM is similar to the one used in KENN: a base NN (or any learner) is used as a foundation which provides initial predictions: on those, the model performs a post elaboration step, enforcing logical rules and improving the predictions. Also, like in KENN, the clause weights are trained together with the whole model, which is a desirable thing in real use cases. However, one significant drawback of RNM is that, for each training step, and also at inference time, an optimization problem must be solved. In comparison, KENN is much more efficient since it is implemented as an actual layer of the NN architecture, which is then trainable end-to-end, together with the clause weights.

3.5 Experiments

In this section we describe in detail the experiments performed with KENN. We tested the ability of KENN of working with relational data, in the context of Collective Classification [43], both with the inductive and transductive learning paradigm.

Definition 3.5.1 (Collective Classification)**.** Consider a directed graph, consisting in a set of nodes V and a set of edges E. Each $v \in V$ is described with a vector of features $x \in \mathbb{R}^n$ and belongs to one of k classes $\{\omega_i\}_{i=1}^k$. The set of nodes V is further divided in two subsets of nodes: X, the set of nodes for which the correct label is known (Training Set), and Y, the set of nodes for which it is unknown (Test Set). The task of Collective Classification is to correctly predict the labels of nodes in Y, given the feature vectors of X and the topology structure determined by E.

Based on how the edges from E are split between Training and Test set, a different learning paradigm is defined:

- **Inductive Learning**: two separate graphs are used, $G_x = (X, E_x)$ for training and $G_y = (Y, E_y)$ for testing, where $E_x = \{(u, v)|u, v \in X\}$ and $E_y = \{(u, v)|u, v \in Y\}$. In other words, the edges of nodes between train and test set are not considered.
- **Transductive Learning**: in the training graph, also edges connecting training and test data are retained. This means that the model can use additional information, in the form of connections between test and training data, even during training.

We also provide a comparison of our results with the ones from SBR and RNM, reported on [44], on the same dataset and tasks.

3.5.1 Citeseer Dataset

The experiments were conducted on the Citeseer Dataset [45]: it consists in a citation network with 4732 citations (directed links) between 3312 scientific publications (nodes), belonging to 6 different classes which represent the topic of the paper. Each node in the dataset is represented by a $0/1$ valued feature vector, where each entry indicates the absence or presence of the corresponding word in the dictionary, which is constituted by 3703 unique words.

3.5.2 The Prior Knowledge

The knowledge that we want to use in order to improve the predictions from the NN is the intuitive fact that, if a paper cites another paper, it is probably true that they are of the same topic. If we denote with $T_i(x)$ the truth value that node x belongs to the i-th output class, this fact can be encoded in terms of a logical clause as follows:

$$\forall x \forall y \quad T_i(x) \wedge \mathrm{Cite}(x, y) \rightarrow T_i(y), \quad i = 1, \ldots, 6$$

meaning that the this clause is repeated one time for each different output class. Inside KENN, this clause is represented as a disjunction of literals as follows:

$$\forall x \forall y \quad \neg T_i(x) \vee \neg \,\mathrm{Cite}(x, y) \vee T_i(y), \quad i = 1, \ldots, 6$$

3.5.3 Results

In Table 3.1 we report the results of the experiments for the inductive case. The accuracies for KENN are the mean of 500 different runs on different randomly extracted splits. In Table 3.2 the results for the transductive case are reported.

Table 3.1: Results of kenn experiments for the inductive case....

% training	NN	SBR	RNM	NN	KENN
10	0.645	0.650 (+0.005)	**0.685** (+0.040)	0.544	0.601 (+**0.048**)
25	0.674	0.682 (+0.008)	**0.709** (+0.035)	0.629	0.671 (+**0.041**)
50	0.707	0.712 (+0.005)	**0.726** (+0.019)	0.680	0.714 (+**0.034**)
75	0.717	0.719 (+0.002)	0.726 (+0.009)	0.733	**0.754** (+**0.021**)
90	0.723	0.726 (+0.003)	0.732 (+0.009)	0.759	**0.768** (+**0.010**)

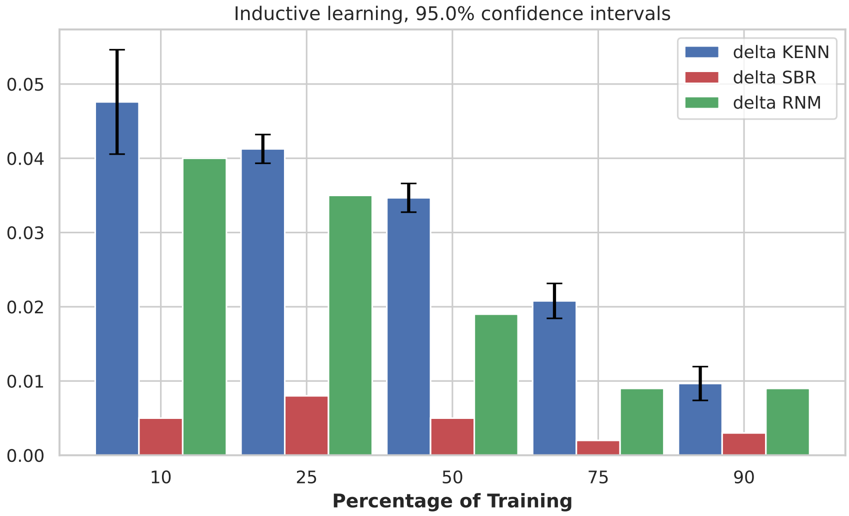


Figure 3.6: Deltas for the inductive learning task. 95% confidence intervals.

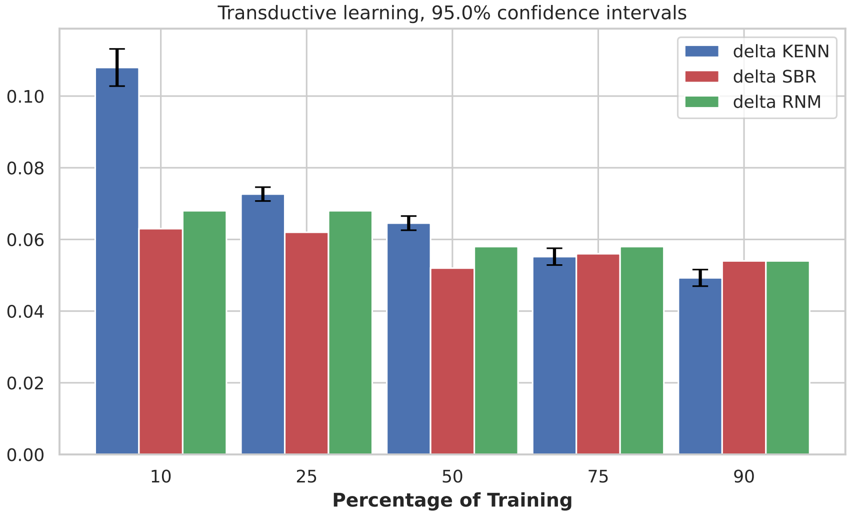


Figure 3.7: Deltas for the transductive learning task. 95% confidence intervals.

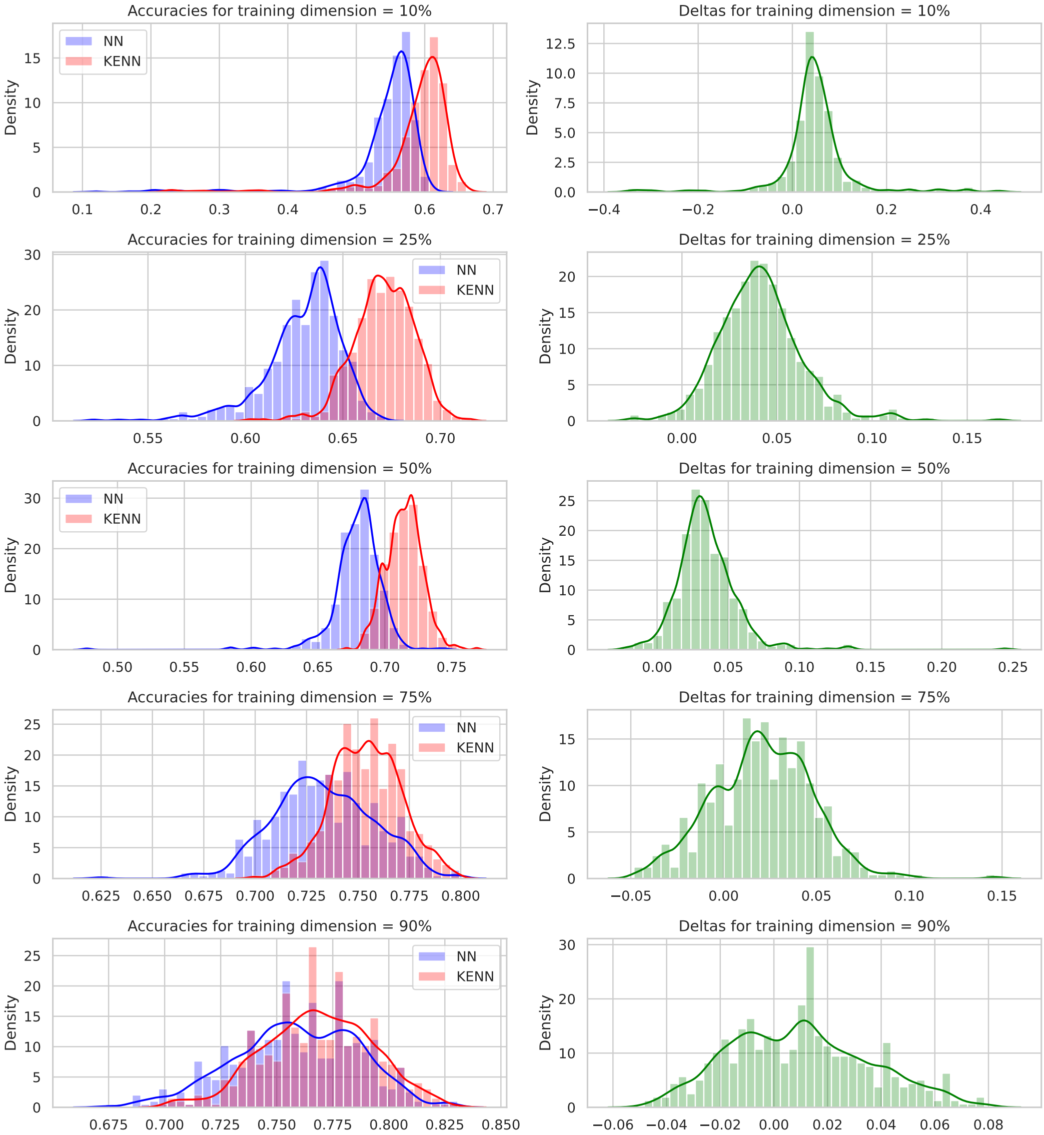
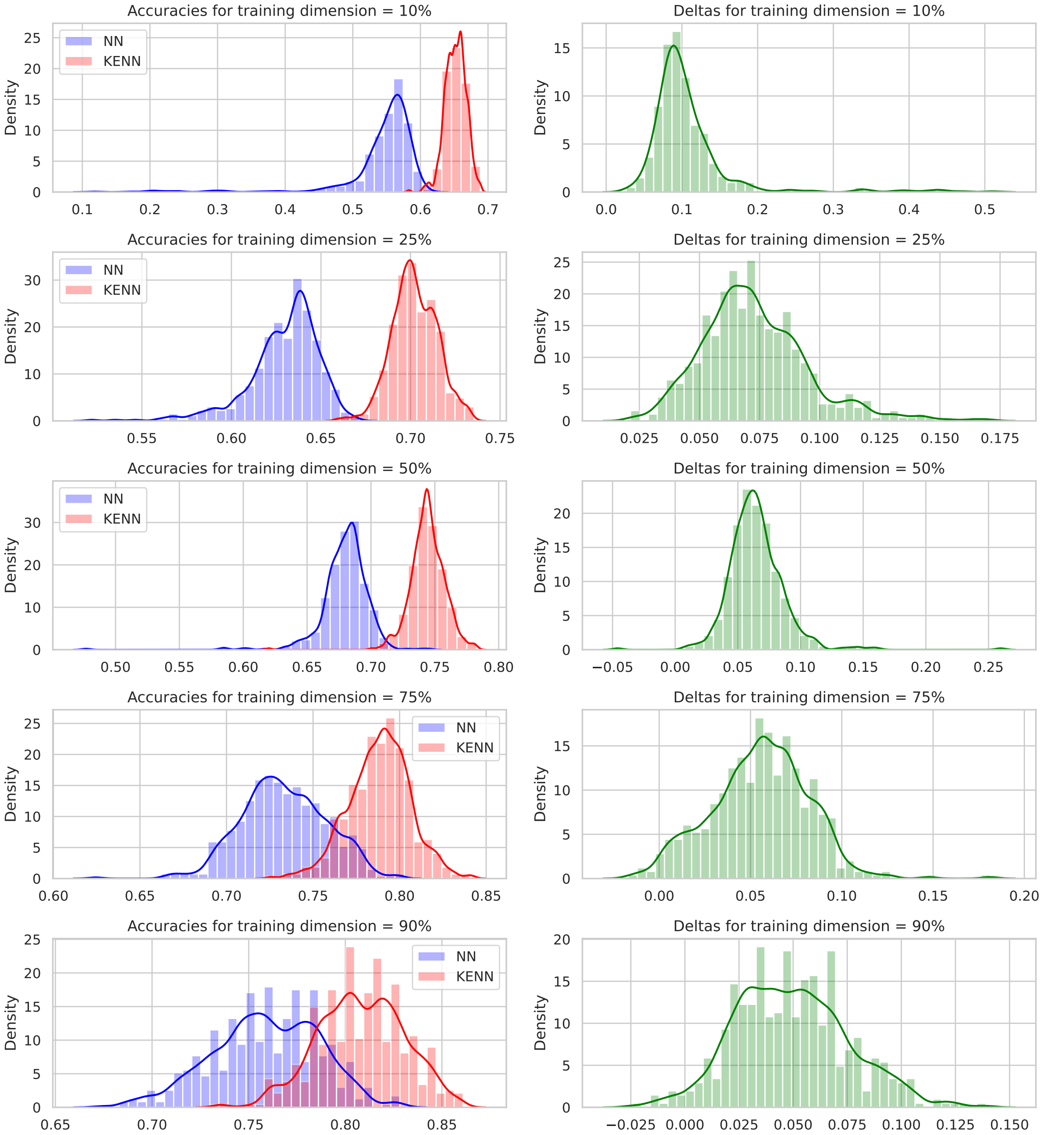


Figure 3.8: Histograms showing the distribution of the accuracies for all the different 500 runs, for the inductive case. On the left, the accuracies of the base NN vs accuracies of KENN. On the right the distribution of the difference between the NN accuracy vs KENN accuracy.


Accuracies for training dimension = 10%

Deltas for training dimension = 10%

Accuracies for training dimension = 25%

Deltas for training dimension = 25%

Accuracies for training dimension = 50%

Deltas for training dimension = 50%

Accuracies for training dimension = 75%

Deltas for training dimension = 75%

Accuracies for training dimension = 90%

Deltas for training dimension = 90%

Figure 3.9: Histograms showing the distribution of the accuracies for all the different 500 runs, for the transductive case. On the left, the accuracies of the base NN vs accuracies of KENN. On the right the distribution of the difference between the NN accuracy vs KENN accuracy.

Table 3.2: Results of kenn experiments for the transductive case....

% training	NN	SBR	RNM	NN	KENN
10	0.640	0.703 (+0.063)	**0.708** (+0.068)	0.544	0.652 (+**0.108**)
25	0.667	0.729 (+0.062)	**0.735** (+0.068)	0.629	0.702 (+**0.073**)
50	0.695	0.747 (+0.052)	**0.753** +0.058)	0.680	0.744 (+**0.065**)
75	0.708	0.764 (+0.056)	0.766 (+**0.058**)	0.733	**0.788** (+0.055)
90	0.726	0.780 (+**0.054**)	0.780 (+**0.054**)	0.759	**0.808** (+0.049)

3.5.4 Clause Weights and satisfaction of the rules

3.6 Explainability in KENN

4 Conclusion

The conclusion goes here.

References

[1] 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

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

[3] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *arXiv preprint arXiv:1710.09829*, 2017.

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

[5] C. Buckner, "Adversarial examples and the deeper riddle of induction: The need for a theory of artifacts in deep learning," *arXiv preprint arXiv:2003.11917*, 2020.

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

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

[8] W. E. Zhang, Q. Z. Sheng, A. A. F. Alhazmi, and C. Li, "Generating textual adversarial examples for deep learning models: A survey," *arXiv preprint arXiv:1901.06796*, p. 129, 2019.

[9] 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.

[10] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a "right to explanation"," *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.

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

[12] 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

[13] S. Bromberger, *On what we know we don't know: Explanation, theory, linguistics, and how questions shape them.* University of Chicago Press, 1992.

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

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

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

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

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

[19] B. Chen, Y. Li, S. Zhang, H. Lian, and T. He, "A deep learning method for judicial decision support," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019, pp. 145–149.

[20] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *arXiv preprint arXiv:1804.11192*, 2018.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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

[27] 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

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

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

[30] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1995–2003.

[31] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the right reasons: Training differentiable models by constraining their explanations," *arXiv preprint arXiv:1703.03717*, 2017.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[35] D. H. Park, L. A. Hendricks, Z. Akata, A. Rohrbach, B. Schiele, T. Darrell, and M. Rohrbach, "Multimodal explanations: Justifying decisions and pointing to the evidence," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8779–8788.

[36] A. Das, H. Agrawal, L. Zitnick, D. Parikh, and D. Batra, "Human attention in visual question answering: Do humans and deep networks look at the same regions?" *Computer Vision and Image Understanding*, vol. 163, pp. 90–100, 2017.

[37] R. Caruana, H. Kangarloo, J. D. Dionisio, U. Sinha, and D. Johnson, "Case-based explanation of non-case-based learning methods." in *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 1999, p. 212.

[38] T. R. Besold, A. Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K.-U. Kühnberger, L. Lamb, D. Lowd, P. Lima, L. Penning, G. Pinkas, H. Poon, and G. Zaverucha, "Neural-symbolic learning and reasoning: A survey and interpretation," *ArXiv*, vol. abs/1711.03902, 2017.

[39] L. G. Valiant, "Three problems in computer science," *J. ACM*, vol. 50, no. 1, p. 96–99, Jan. 2003. [Online]. Available: https://doi.org/10.1145/602382.602410

[40] A. Daniele and L. Serafini, "Knowledge enhanced neural networks," in *PRICAI 2019: Trends in Artificial Intelligence*. Cham: Springer International Publishing, 2019, pp. 542–554.

[41] V. Novák, "First-order fuzzy logic," *Studia logica*, vol. 46, no. 1, pp. 87–109, 1987.

[42] L. Serafini and A. d. Garcez, "Logic tensor networks: Deep learning and logical reasoning from data and knowledge," *arXiv preprint arXiv:1606.04422*, 2016.

[43] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[44] G. Marra, M. Diligenti, F. Giannini, M. Gori, and M. Maggini, "Relational neural machines," *arXiv preprint arXiv:2002.02193*, 2020.

[45] Q. Lu and L. Getoor, "Link-based text classification," in *IJCAI Workshop on Text Mining and Link Analysis*, 2003.

Acknowledgments

This is the acknowledgments section.