Università degli Studi di Milano Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Neuro-Symbolic Fine-grained Entity Typing with KENN

Relatore: *Prof. Matteo Palmonari*
Co-relatore: *Dott. Manuel Vimercati*

Tesi di Laurea Magistrale di:
Christian Bernasconi
Matricola 816423

Anno Accademico 2020–2021

Abstract

Deep Neural Networks (DNNs) in the field of Natural Language Processing (NLP) have proved to be of primary importance due to the significant benefit they can provide to a great diversity of applications. The NLP task of interest of this thesis is Entity Typing (ET), which aims to assign types from a type set to named entities appearing in a text, and it is beneficial for several NLP applications such as Question Answering, Entity Linking, Relation Extraction, and many others. When the type set has a high cardinality, this task takes the name of Fine-grained Entity Typing (FET). The type set of a FET dataset is usually organized in a hierarchical way, where the relations between coarse types and fine-grained types are represented by the edges of a tree. The main goal of this work is to study Neuro-Symbolic Integration (NSI) techniques to exploit hierarchical relations through the use of logical knowledge. Among the available NSI approaches, KENN is the framework chosen to conduct the experiments. KENN is a novel approach that seems suitable for our purpose since it allows injecting logical knowledge into a DNN in a very straightforward way. The focus of this thesis is to adapt KENN to the specifics of a FET problem by exploiting hierarchical information at its best. In particular, the experiments involve two popular FET datasets: BBN and FIGER.

Contents

1 Introduction

1.1 Motivation and background

Deep Neural Networks (DNNs) in the field of Natural Language Processing (NLP) have proved to be of primary importance due to the significant benefit they can provide to a great diversity of applications that spans from text translation, summarization, or information extraction, among others. Such approaches are known to be data greedy, thus requiring a considerable amount of annotated data. Unfortunately, building an annotated dataset is an expensive and sophisticated activity, since the annotated data have to be as precise as possible according to the task of interest. In addition, the information they carry about the real world can easily become obsolete because it inherently changes over time or just becomes more complex. A task that has become more complex over time is Entity Typing (ET). Originally proposed by [1] as a subtask of Named Entity Recognition (NER), the first type set was composed of only three types: `Person, Organization,` and `Location`. However, the last decades have experienced an increasing interest in the task, since information about named entities can be exploited to improve several downstream tasks such as Question Answering, Entity Linking, Relation Extraction, and many others. Along with expanding the type set, here emerges the need to organize the types in structures that account for specialization, thus obtaining a hierarchy (e.g., [2, 3] used subtypes of `Person` and `Location`). Concurrently, several datasets have been proposed [4, 5, 6] introducing a high number of new types by providing a specialization or adding branches to the initial hierarchy. When dealing with datasets having such type sets, the ET task goes under the name of Fine-grained Entity Typing (FET). Given these premises, it is possible to outline a real-case scenario in which a DNN for FET may need to learn new types that were unknown at training time.

When a DNN model for FET has to be adapted to new types, it may be convenient to draw upon the knowledge that has already been acquired to favor the adaptation process. This operation can be carried out by defining relations between learned types and new types through the use of external knowledge, such as knowledge bases and/or domain experts. We can identify two relations that can help in the discrimination of new types: specialization and disjointness. Specialization describes the subtype relation between two types (e.g., `Actor` is the specialization of `Person`), disjointness describes mutual exclusivity on entities between types (e.g., if an entity is labeled as `Person` it cannot be labeled as `Location` and vice versa). Specialization and disjointness can be easily mapped to the disposition of types in a hierarchy: the former corresponds to the is-a relation between a subtype and a supertype, the latter expresses mutual exclusivity between types from different branches.

The specialization and disjointness relations can be induced by a DNN during the training of new types. However, even if the is-a relation is given by the presence of the *supertype* on each occurrence of the *subtype* in a dataset, a DNN could still assign the *subtype* without assigning the *supertype*, thus leading to

a wrong type assignment. Disjointness is even more complex to induce since it is difficult to make assumptions on what a DNN can learn about the co-occurrence of types from different branches. Since the induction process is not obvious, having a technique to inject external knowledge into a DNN could be helpful to guide the learning process.

Neuro-Symbolic Integration (NSI) approaches have shown to be useful in multiple tasks through the peculiar characteristics of mixing neural representation and symbolic knowledge. In particular, different approaches based on the integration of explicit knowledge have been proposed during the last years [7]. Using an NSI approach to explicitly model and inject external knowledge into a neural ET approach may be a valid solution to face the problem of adapting a trained model to the presence of new types. If so, injecting external knowledge about the type dependencies discussed above can lead to a promising scenario: since it will be no more necessary to induce knowledge from data, the training would benefit in terms of time and effectiveness thanks to the external knowledge. For example, let us assume that the new type `Politician` has to be learned by a DNN: if we know that `Politician` is a subtype of the already learned type `Person`, we can use the prediction of `Person` to influence the prediction of `Politician`, thus speeding up the learning process.

Starting from the assumption that in specific scenarios like FET, exploiting prior knowledge may actually accelerate the training process by requiring less training data, particularly when dealing with underrepresented, specific, or even unseen types, this thesis aims to inspect the usage and behavior of a literature NSI approach: Knowledge Enhanced Neural Networks (KENN) [8]. The main reasons that led to the choice of KENN among other state-of-the-art solutions are the following:

1. It allows to define the hierarchical relations of interest through logical clauses
2. It can be placed on top of any DNN to influence its predictions, thus allowing to observe how a model behaves in presence of different configurations of KENN
3. The effect of each logical clause can be analyzed in detail to understand the importance of that clause
4. Experiments in [8, 9] showed that KENN is suitable for zero-shot learning scenarios, so it can be a promising solution to face the problem of the adaptation of a trained model to the presence of new types

1.2 Research questions

Now that the context of this thesis has been illustrated, we can sum up the goal of this work with the following research questions:

- ***RQ1:*** *How can we express hierarchical information through logical rules?* → define a strategy to translate the relations of specialization and disjointness into logical rules

- ***RQ2:*** *How does KENN behave with different configurations?*
  → define a parameter space to test KENN with different setups
  → find the most promising parameters configuration

- ***RQ3:*** *What are the benefits of using KENN in FET?*
  → evaluate whether the injection of prior knowledge may accelerate the learning process, impact the performance or, more generally, provide additional benefits to a DNN for FET

1.3 Outline

Section 2 provides the background knowledge of the topics involved in this thesis. After an overview of Language Models, Entity Typing, and Fuzzy Logic, this section introduces Neural-Symbolic Integration and provides a comparison between KENN and other approaches of interest from the literature.

In section 3, the architectural details and language of KENN are described by focusing on the most relevant aspects. Then, several strategies to define logical clauses starting from a hierarchy will be proposed within some preliminary experiments.

Next, in section 4 we can find the description of the datasets used for the evaluation, followed by the architecture of the adopted models.

Proceeding to section 5, which is the core of this thesis, it contains all the experiments carried out. Each experiment is presented by providing the motivation and the experimental setup. Since the setup of some experiments may derive from previous results, each experiment will be immediately followed by a brief discussion on the outcome.

Finally, we have section 6, which sums up the conclusions of each experiment to answer the research questions one by one, and section 7, which proposes some promising future works.

2 State Of The Art

2.1 Language Models

Since the Entity Typing task deals with textual features, we need to introduce some basic concepts regarding text representation. Over the years, a lot of approaches have been proposed to transform the text into a machine-readable format, reaching more and more effectiveness in several downstream Natural Language Processing (NLP) tasks. A wide variety of language models is available, providing different ways to represent textual information in a vectorial space. Choosing the right language model is crucial when facing NLP problems and could be decisive to achieve quality results: if we start from a low-quality text representation, we cannot expect high-quality results for any following task [10].

Before proceeding with the overview, it is good to provide a general definition of language model: *a language model is a statistical model which represents probability distributions over sequences of words.* Given a sequence of words $W = w1, ..., w_n$, a language model is able to compute $P(W)$ or $P(w_{n+1}|w_1, ..., w_n)$.

2.1.1 Overview

The first approaches were based on the frequencies of the words in a corpus. The classic example is the Bag Of Word (BOW) model. Introduced by Zellig Harris in 1954 [11], it has become the starting point of many future works. This approach has lots of limitations, especially the sparsity of its high-dimensionality vectors. To overcome these limits, other valid solutions have been proposed through the years under the name of Word Embeddings. The common goal of these newer approaches is to map the words of a vocabulary from a high-dimensionality space into a real vector space with low-dimensionality. Word Embeddings rely on the distributional hypothesis for which similar words tend to appear in similar contexts. These models can be grouped into two main categories: count-based models and predictive models.

One of the most popular count-based approaches is GloVe [12]. It captures the statistics of a corpus by computing word-word co-occurrences, then for each pair of words it optimizes an objective that relates co-occurrences to contexts.

Moving on to predictive models, we can find the famous Skip-Gram (SG) and Continuous Bag Of Words (CBOW) methods. Best known as Word2Vec, they are based on neural networks. For both models, the embedding vectors correspond to the learned weights of the hidden layer of the network. The aspect that distinguishes the two approaches is that while CBOW is trained to predict the central word of a sliding window given the context words, SG is trained to predict context words given the central word. With both methodologies, it is possible to learn high-quality word vectors maintaining low-dimensionality, where vector similarity can be measured using the cosine similarity. The more two words have similar meanings, the closer they will be in the vector space.

In 2017 the advent of transformers had an important impact in the NLP

research area. The publication [13] introduced this new neural architecture based on encoders and decoders. Transformers rely on a sequence-to-sequence architecture and were originally used for machine translation. Differently from Recurrent Neural Networks (RNN), which operate on sequences and are intrinsically sequential, transformers operate on sets in a parallel way. Instead of using a single compressed encoding coming from the last encoder like in RNN, in transformers the decoders can access the information of each input element. The core of these models is the Self-Attention mechanism that is applied to the input tokens of each transformer encoder/decoder. This operation makes the model able to capture deeper contextual information and dependencies, thus providing a much better representation than the previous approaches. Since the attention operation works on sets and not on sequences, the model can be constrained by adding positional encodings to deal with text sequences. Transformers architecture inspired future works like BERT [14] and GPT [15], which gained more and more popularity until nowadays. They both exploit a mechanism based on Self-Attention, but while the first is composed of a stack of transformer encoders, the latter is constituted by a stack of transformer decoders.

2.1.2 BERT

Inspired from transformers, Devlin et al. presented BERT (Bidirectional Encoder Representations from Transformers) in 2019 [14]. Unlike the original transformer architecture, it does not use any decoder. The fact that it uses a stack of 12 bidirectional transformer encoders makes BERT able to exploit both the left and right context of each word. Moreover, thanks to the attention mechanism inherited from transformers, this architecture can be used to generate very powerful language models with dense neural representation.

Before feeding the model, the input text is transformed into a sequence in the form of WordPiece embeddings [16] with a vocabulary counting 30,000 tokens. Using this representation, each word is decomposed into subwords, thus allowing to improve the handling of rare words. The format of an input sequence can contain special tokens like `[CLS]` and `[SEP]`. The first one is the *classification token* and its final hidden state can be used as the aggregate sequence representation for classification tasks. The latter stands for *separator token* and can be used when we need to differentiate the sentences which constitute the input.

BERT earned most of its popularity thanks to the fact that it can be used in a wide variety of NLP downstream tasks without big efforts. Indeed, it comes as a model pre-trained on a large amount of data and provides a generic language representation that can be fine-tuned to face any problem (e.g., Question Answering, classification, NER, etc.). A sketch of the training procedure is shown in Figure 1. The pre-training procedure is performed using two strategies:

1. **Masked Language Model:** for each input sequence, the 15% of tokens are replaced by the `[MASK]` token. Then, the model is trained to predict the original values of the masked tokens by exploiting the left and right context.

2. **Next Sentence Prediction (NSP):** each input sequence is split into two parts separated by the `[SEP]` token. Then, the 50% of the sequences get modified by replacing the second part using a random one from another sequence. Finally, the model is trained to distinguish positive and negative sequences.

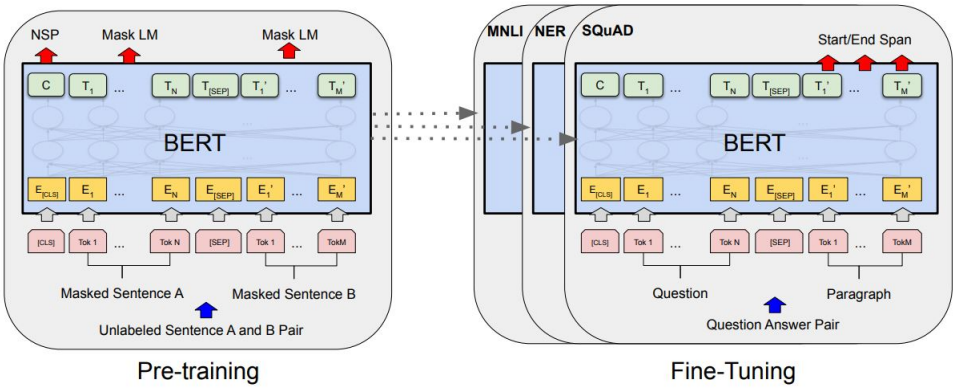


Figure 1: BERT's pre-training and fine-tuning. (Source: [14])

Until now, we only talked about the strengths of BERT. However, the high quality of the text representation is achieved at the expense of high resources usage. BERT's model comes originally in two versions: BERT base (110M parameters) and BERT large (340M parameters). These model sizes require high memory capabilities and make the training procedure not accessible to everyone. For this reason, different new versions have been proposed over the last few years and several examples can be found at HuggingFace[1]. Among these versions, we can find BERT-based models which are lighter than the original (e.g., DistilBERT, RoBERTa, etc.) and models fine-tuned for specific tasks or different languages.

2.1.3 DistilBERT

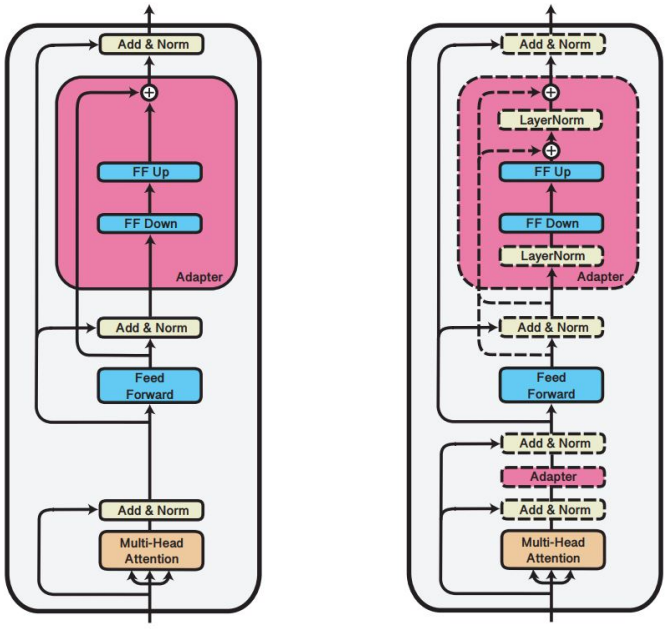
As suggested by the name, DistilBERT is a distilled version of BERT [17]. With 40% fewer parameters, it is 35% faster at inference time and it is effective at 97% with respect to the performance of the original version of BERT. Thanks to its lighter architecture composed of only 6 transformer encoders, it can be fine-tuned in a more reasonable time. DistilBERT model is pre-trained using knowledge distillation. This procedure is a compression technique in which a smaller *student* model (i.e., DistilBERT) is trained to emulate a larger *teacher* model (i.e., BERT).

[1] https://huggingface.co/models

2.1.4 Adapters

The fine-tuning of a model requires less time than training it from scratch. However, even this procedure can be quite expensive, especially when dealing with very large models. An alternative solution to adapt a BERT-based model to a new task can be found in the *adapters* [18, 19]. The key concept behind the adapters is to introduce a small set of learnable parameters for each transformer layer. In the case of BERT and its variants, these parameters are added to each transformer encoder. The training is then performed by freezing all the weights of the pre-trained language model and updating only the parameters introduced by the adapters. As shown in Figure 2, the new parameters can be added following two strategies:

- **Pfeiffer:** an adapter is added after each transformer encoder (Figure 2a)
- **Houlsby:** two adapters are added for each transformer encoder, one after the attention layer and one after the feedforward layer (Figure 2b)



(a) Pfeiffer strategy (b) Houlsby strategy

Figure 2: Adapters architectures. (Source: [19])

The modifications introduced by both the architectures can be seen as the addition of auto-encoders to the model. The dimensionality of the input is reduced (with respect to a customizable reduction factor) through a first feed-forward layer (FF-DOWN in Figure 2) and then expanded to its original size

through another feedforward layer (FF-UP in Figure 2). In this way, a pre-trained model can adapt to new tasks without requiring any fine-tuning of the original weights. In addition, Pfeiffer et al. show in [19] that their approach outperforms traditional strategies such as full fine-tuning as well as multitask learning.

2.2 Entity Typing

2.2.1 Overview

Entity Typing (ET) is a subtask of Named Entity Recognition (NER) originally proposed in the 90s [1]. ET aims to assign types[2] from a type set to named entities appearing in a text. Given a type set T, a sentence $S = c_L + m + c_R$ where m is the mention span and $c_{L/R}$ is the left/right context, the goal of ET is to predict the correct types $t_m \subseteq T$ that better describe m with respect to S. Since t_m may contain multiple types, the ET task is considered a multiclass multilabel classification problem [20].

Figure 3 shows the steps involved in a generic problem of ET. The input sentence requires to classify the mention of "George Washington" with respect to the given context. Before feeding any machine learning model, the text must be transformed into a machine-readable format. This step can be based on the extraction of handcrafted features or, for most approaches from recent years can rely on newer encoding strategies ranging from Word2Vec to BERT. Finally, the preprocessed input is provided to the model that produces an output vector containing the predictions for each type in T. Depending on the adopted model, ET approaches can be grouped into two main categories: classification approaches like [21, 4, 22, 23, 24], or regression approaches like [25, 26, 27, 28].

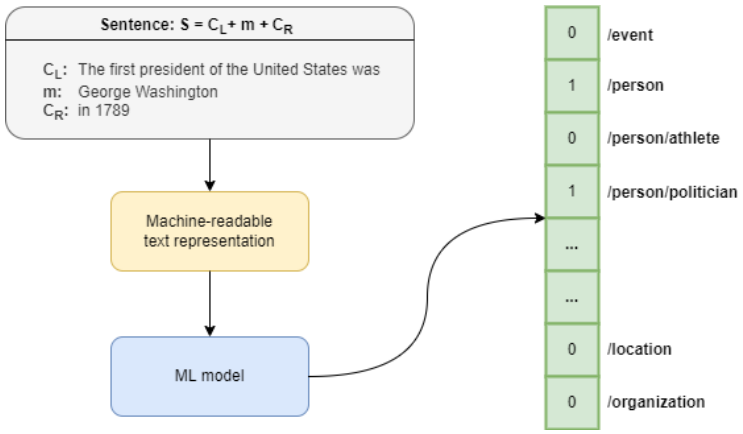


Figure 3: Entity typing pipeline

Using ET to detect the semantic classes of named entities can be beneficial

[2]synonyms that could be used in this work: labels, classes

for several NLP applications such as Question Answering, Knowledge Base Population, Relation Extraction, Entity Linking, and many others. Sometimes, the context of an NLP application requires dealing with very specific types, thus needing the extension of the typical type set composed of `Person`, `Organization` and `Location`. For this reason, several datasets with larger type sets have been proposed over the years and used to evaluate such tasks [4, 29, 23, 6]. When the type set has a high cardinality, the ET task takes the name of Fine-Grained Entity Typing (FET). The type set of a FET dataset is usually organized in a hierarchical way, where the relations between coarse types and fine-grained types are represented by the edges of a tree. Each coarse type can be seen as the root node of a tree. Regarding the fine-grained types, they are represented as the descendants of the root: the deeper the node, the finer the type. The relations expressed by these hierarchical trees are the type dependencies we want to exploit in this thesis using Neural-Symbolic Integration.

2.2.2 Approaches that exploit hierarchical information

As discussed in the introduction, exploiting the hierarchy is a technique used by different approaches to face FET tasks. In [30] a model is trained to predict one branch of the hierarchy and one model per branch is trained to specialize the classification, obtaining 30 models. Similarly, [21] defines a classifier for each type and uses them following the hierarchy in a top-down manner.

[25, 31, 28] defines a single classifier for all types, but filter the predictions using the hierarchy in a top-down manner, filtering out the types that are not descendants of already predicted types. The differences between these approaches reside in the inference technique: [25] uses a fixed threshold; [31] a relative threshold; [28] infers the type with the maximum logit along with its brothers, then processes the sons of the inferred type until the bottom of the hierarchy is reached or the maximum is lower than a fixed threshold. The inference method of [28] is used also by [26, 32].

[31, 28, 33, 34, 24, 27, 35, 32, 36] inject hierarchical information in the model through type representation and/or optimization techniques. [31, 33] represent each type through a one-hot vector with values 1 for the type and for its ancestors. [28, 35] use hierarchy to derive similarities between types, thus weighting the errors. [34] represent each type with a vector obtained with Rescal [37] or ComplEx [38], thus injecting also hierarchical information about types. [24] boost the prediction of a node based on its ancestors. [27, 32] use a layer to predict types for each level in the hierarchy, then each layer uses the higher level prediction as input. [36] use the hierarchy to select hard negatives for a learning-to-rank training technique.

Details on how hierarchical rules are exploited in this work will be described in section 3.2.2.

2.2.3 Approaches based on BERT

In the architecture of a FET network, the encoding of the entity mention within its context is a central characteristic of each approach. Since in our architecture BERT is central for the encoder, we now focus on the approaches that use BERT or derived language models as encoder. [39] tested different contextual encoders by fixing an architecture and showed that BERT-based encoders achieved better performance in FET. Similarly, [40, 41, 42] use BERT as encoder to obtain a contextual vector from the input mention within its context. [43] use BERT to obtain types by masking the entity mention in the input sentence and optimizing the generated tokens to obtain the tokens that represent the correct types. Finally, [44] uses an ensemble encoder comprising SpanBERT [45].

Details on the FET network architecture designed for the experiments of this thesis will be presented in sections 4.2.

2.3 Fuzzy Logic

Fuzzy Logic (FL) aims at modeling the human reasoning system. It finds its roots in multi-valued logic, which extends the classical two-valued logic by adding new truth values. Differently from other multi-valued logics which have a finite number of truth values, in FL a variable can assume the value of any real number in [0, 1], where 0 means completely false and 1 means completely true. If we think about the definition of logic, it *"is the science of the formal principle of reasoning"*. From this perspective, FL is concerned with the formal principles of *approximate reasoning*, since we deal with imprecise premises from which we derive imprecise conclusions [46]. This aspect reflects the fact that in human reasoning we have to deal with vagueness when classifying concepts such as a *tall person* or a *small number* [47, 48]. In this work, we will focus on the fuzzy extension of the First Order Logic (FOL).

Several FL implementations have been proposed over the years to approximate the logical operators and quantifiers in different ways. We now proceed with the description of some popular variants of the FL by assuming familiarity with the basic notions of the FOL. The definitions we will use in this section are taken from [49]. First of all, the semantics of the main fuzzy operators relies on the following functions:

- **Fuzzy Negation:** function to compute the *negation* of a truth value of a formula. A *fuzzy negation* is a decreasing function $N : [0, 1] \to [0, 1]$ such that $N(1) = 0$ and $\forall(x) \in [0, 1], N(N(x)) \geq x$. N is called *strict* if it is strictly decreasing and continuous, and *strong* if $\forall x \in [0, 1], N(N(x)) = x$.

- **Triangular Norms (t-norms):** function to compute the *conjunction* of two truth values of a formula. A *t-norm* is a function $T : [0, 1]^2 \to [0, 1]$ that is *commutative* and *associative*, where $\forall x \in [0, 1], T(x, \cdot)$ is increasing (*monotonicity*) and $\forall x \in [0, 1], T(1, x) = x$ (*neutrality*).

Table 1: Some common t-norm and t-conorm in fuzzy logic

Name	**T-norm**	**T-conorm**
Gödel	$T_G(a,b) = min(a,b)$	$S_G(a,b) = max(a,b)$
Product	$T_P(a,b) = a \cdot b$	$S_P(a,b) = a + b - a \cdot b$
Łukasiewicz	$T_L(a,b) = max(a+b-1,0)$	$S_L(a,b) = min(a+b,1)$

- **Triangular Conorms (t-conorm):** function to compute the *disjunction* of two truth values of a formula. A *t-conorm* (also known as s-norm) is a function $S : [0,1]^2 \to [0,1]$ that is *commutative* and *associative*, where $\forall x \in [0,1], S(x,\cdot)$ is increasing (*monotonicity*) and $\forall x \in [0,1], S(0,x) = x$ (*neutrality*). Since t-conorms are obtained from t-norms by applying De Morgran's law from classical logic, it is possible to write the following equivalence

$$S(a,b) = 1 - T(1-a, 1-b)$$

- **Aggregation Operators:** function to compute $\forall$ and $\exists$ quantifiers.
  - the $\forall$ quantifier can be interpreted as a function $A : [0,1]^n \to [0,1]$ computed as a conjunction over all arguments x, using a t-norm adapted to n-dimensional input:

$$A_T() = 0$$
$$A_T(x_1, x_2, ..., x_n) = T(x_1, A_T(x_2, ..., x_n))$$

  - the $\exists$ quantifier can be interpreted as a function $E : [0,1]^n \to [0,1]$ computed as a disjunction over all arguments x, using a t-conorm adapted to n-dimensional input:

$$E_S() = 0$$
$$E_S(x_1, x_2, ..., x_n) = S(x_1, E_S(x_2, ..., x_n))$$

Starting from these definitions, we can find a variety of implementations of fuzzy operators. In Table 1 are reported the operators of some popular fuzzy logics. The variants proposed by Łukasiewicz and Gödel can be considered among the most popular, but there are many other interpretations (e.g., Weber, Yager, Fodor). Table 2 reports some aggregation operators based on the presented t-norm and t-conorm operators.

2.4 Neuro-Symbolic Integration

2.4.1 Symbolic vs non-symbolic

In the last decade, the use of Deep Neural Networks (DNN) gained popularity in any machine learning context with very promising results. However, the need

Table 2: Some common aggregation operators in fuzzy logic

Name	Generalizes	Aggregation operator
Minimum	T_G	$A_{T_G}(x_1, ..., x_n) = min(x_1, ..., x_n)$
Product	T_P	$A_{T_P}(x_1, ..., x_n) = \prod_{i=1}^{n} x_i$
Łukasiewicz	T_L	$A_{T_L}(x_1, ..., x_n) = max(\sum_{i=1}^{n} x_i - (n-1), 0)$
Maximum	S_G	$E_{S_G}(x_1, ..., x_n) = max(x_1, ..., x_n)$
Probabilistic Sum	S_P	$E_{S_P}(x_1, ..., x_n) = 1 - \prod_{i=1}^{n}(1 - x_i)$
Bounded Sum	S_L	$E_{S_L}(x_1, ..., x_n) = min(\sum_{i=1}^{n} x_i, 1)$

for better soundness and trustworthiness, as well as a better understanding of neural models, led the research community to investigate new methodologies to bring these aspects into the deep learning world. Several new methods to integrate knowledge and DNN have emerged in recent years with this ambition. This line of research is called *Neuro-Symbolic Integration* (NSI) and it represents a hot topic in the AI community [50]. The name combines the terms *neural* and *symbolic*, which respectively refer to neural networks and symbolic reasoning. More in detail, this research area aims to take benefits from both these two subfields of AI to support each other and overcome their limitations.

Starting from DNNs, it is well known they need a big amount of labeled training examples to generalize well. Unfortunately, such quantities of annotated data are not always available due to the big effort needed to be produced. Another limitation is that they lack interpretability because the learned functions are seen as not explainable black-boxes by humans. Furthermore, an issue that may occur when dealing with DNNs is the distributional shift (i.e., the different distribution between the training examples and the real-world data). Moving on to symbolic approaches, they do not suffer from the previous problems. Indeed, they can easily generalize through logical rules without the need for large data. Moreover, they are explainable thanks to the high-level interpretability of the logical knowledge provided by a domain expert. However, concerning DNNs, these kinds of approaches are heavily influenced by noise and are not as efficient as the neural ones.

Depending on the task for which it is designed, an NSI system can combine the symbolic and non-symbolic elements in several ways as we will see in the next section.

2.4.2 Neuro-Symbolic Integration approaches

The first boundary between NSI approaches can be drawn with respect to the field for which they are designed. We can identify two main areas where these approaches are applied: machine learning and symbolic reasoning. In the first case, the integration aims to regulate the learning process with the use of logical knowledge. The goal of this family of approaches is to exploit the computational efficiency of DNNs and the expressiveness of logic to improve neural model capabilities in classical machine learning tasks, ranging from classification (e.g., binary, multiclass, multilabel, collective) to embedding learning, also covering

relational learning tasks. On the other hand, we can find NSI approaches that are designed to solve problems from the symbolic field. In this second family of approaches, a DNN is trained to learn and perform symbolic reasoning over knowledge. The target of these works is more about Inductive Logic Programming (ILP) tasks and related.

Another possible distinction between NSI works concerns the adopted logical language. Looking at the approaches proposed in the literature, we can notice that there is not a standard way to represent logical knowledge. In most cases, a set of FOL rules is used to set constraints on data. However, a lot of approaches relying on this kind of rules apply some restrictions to the language, thus limiting its expressiveness. Such restrictions may involve the arity of predicates, functions, and logical operators. Other limitations may be imposed on the form that must be used to express logical rules (e.g., Horn clauses). Furthermore, additional assumptions can involve the usage of variable quantifiers (e.g., universal assumption). Most of these restrictions do not affect the approaches based on propositional logic. However, propositional logic is less expressive and flexible than FOL by definition.

Even the semantics of the logical language can differ a lot from one approach to another. If the goal is to have an end-to-end fully differentiable architecture, then it will be necessary to move away from finite-valued logic. The motivation behind this need derives from the nature of neural networks, which deal with continuous real values. For this reason, the most frequent solution is to relax the logic by adopting fuzzy/probabilistic semantics. In this way, it is possible to evaluate logical formulas grounded with values coming from a neural network. The main benefit that derives from the use of infinite-valued logic is that it allows to efficiently train a Neuro-Symbolic model using back-propagation algorithms. As discussed in section 2.3, there are many possible choices between fuzzy operators. Each of them behaves in different ways when used in a differentiable learning setting. In [49] several experiments were performed to show the weakness and strengths of different fuzzy operators.

After this NSI overview about logical language and semantics, we can go deeper and categorize the state-of-the-art works. As proposed in [51], a reasonable way to group the novel approaches is to consider the methodology adopted to inject logical knowledge into the learning process. In Figure 4 we can see three main groups. Each of them is based on different architectures.

Neural architectures for logical reasoning: [52, 53, 54, 55, 56, 57, 58, 59, 60, 61]. These approaches use neural networks to perform probabilistic inference on logical theories by learning reasoning strategies. This category focuses mainly on ILP (e.g., theory learning, theory compression, logical rule induction) and structured data-related tasks (e.g., knowledge graph completion, knowledge graph reasoning, triple classification). The proposed methodologies reach their goals in various ways.

Starting from [57], we find Probabilistic Logic Neural Networks. The proposed models are defined to perform knowledge graph reasoning. A Markov Logic Network [62] is used to compute the joint distribution of all triplets, then

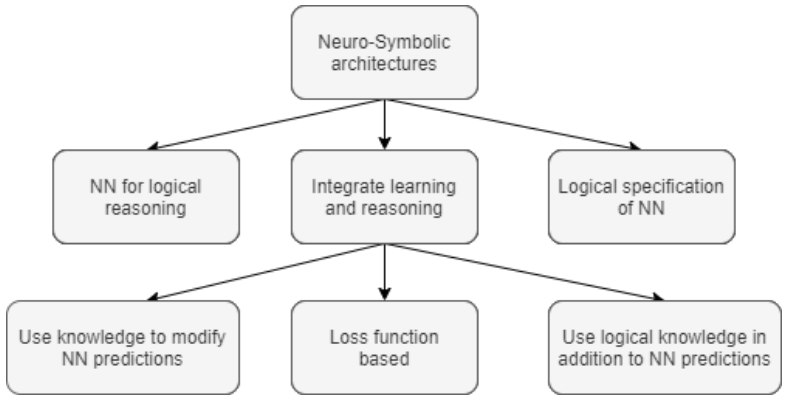


Figure 4: Categorization proposed in [51] to group Neuro-Symbolic approaches based on the knowledge injection methodology

a model is trained with a variational expectation-maximization algorithm. Another approach based on Markov Logic Networks is [56], in which a statistical relational learning system is proposed to learn the implicit representation of FOL rules through neural networks.

Going on, we find [58], [55] and [52]. All of them introduce some FOL language restrictions, supporting only unary and binary predicates. In [58], Hohenecker and Łukasiewicz introduce Recursive Reasoning Networks. In this approach, each training example is a KB composed of different facts. Models are trained on these KBs using a shared ontology, then generate embeddings of individuals. Moving on to [55], Campero et al. propose a task-dependent loss function. Their methodology relies on forward chaining and soft unification to perform inference. Then there is TensorLog [52], where a subset of FOL called *ptree-SDKGs* is provided to represent knowledge in a highly scalable framework for tuning parameters of probabilistic logic.

To conclude with the last three approaches of this family, we find the methodology proposed by Cohen et al. [53], Conditional Theorem Provers (CPT) [54], and SATNet [60]. Cohen et al. propose *sparse-matrix reified KBs* to represent subjects, objects, and relations of a KB. Through these three sparse matrices, they provide an effective and scalable way to represent symbolical knowledge in neural contexts. Moving on to CPT, Minervini et al. propose a method to boost the efficiency of a model in a particular way: during the proving mechanism, the learning process aims to dynamically select only a minimal subset of rules depending on the current goal. At last, we have SATNet by Wang et al.. In this approach, a MaxSAT solver layer is used to make DNNs able to learn logical structures without explicitly specifying any logical rules. Both CPT and SATNet adopt a logical language that is function-free.

Logical specification of neural network architectures: [63, 64]. In these novel approaches, a logical language is used to specify the architecture of neural

networks. This peculiarity leads to better interpretability of DNNs.

In [63], Sourek et al. propose Lifted Relational Neural Networks. These neural models are composed of multiple feed-forward networks (one per example) with shared weights, where each node represents a weighted clause. The methodology proposed in Logical Neural Networks [64] is quite different. In this case, a neural network is modeled as a graph representing the syntax tree of logical formulas. The knowledge is expressed through a function-free FOL language with *weighted nonlinear logic* semantics.

In both approaches, clause weights are learnable parameters. This feature makes the model able to give greater importance to the more relevant rules at prediction time.

Neuro-symbolic architectures for the integration of inductive learning and deductive reasoning: The goal of this group of approaches is to integrate neural and logical components into a unique (possibly fully differentiable) framework. In this way, the processes of learning and reasoning of the two components influence each other to enhance the quality of predictions. Unlike the previous categories, these works are more oriented to exploit logical knowledge in classical machine learning tasks. This family of architectures can be further divided into three subclasses:

- ***Logical knowledge to modify neural networks predictions:*** [65, 66, 67, 8]. These works rely on additional components (e.g., neurons, layers, modules) responsible for handling the logical knowledge.

  Li and Srikumar [65] propose a framework to introduce constraints as logical statements without adding any extra learnable parameters. The base neural model is augmented with *named neurons* corresponding to the predicates of the logical clauses. The post-activation value of a named neuron can be seen as the truth value of the mapped predicate. About the language, rules must be expressed as logical implications with some restrictions: the antecedent is a conjunction/disjunction of literals and the consequent is a single literal.

  The second approach of this family is the one proposed by Daniele and Serafini with Knowledge Enhanced Neural Networks (KENN) [8]. The logical knowledge is injected through an additional layer called *Knowledge Enhancer*. The initial predictions of the base neural network are modified through this layer to increase the satisfaction of each clause by applying a boost function. The logical language is a restricted function-free FOL: operators are limited to disjunction and negation, predicates must be at most binary, and variables are universally quantified.

  Proceeding with the last two approaches, we have Deep Logic Models (DLM) [67] and Relational Neural Machines (RNM) [66] that are quite similar in the way they work. In both cases, the process to compute the output is performed in two phases: a low-level stage to process the input and a semantic stage to perform high-level reasoning. The base neural

network is supported by an undirected graphical model that represents the probability distribution. The two components are combined as follows: given the output of the neural network and the set of logical rules, a maximum a posteriori estimate (MAP) is performed to find the most probable assignment to the grounding atoms. The authors of RNM point out that their approach overcomes some limitations of DLM in terms of scalability and applicability. Furthermore, they promise a tighter connection between the trainer and the reasoner.

- ***Loss function based:*** [68, 69, 70, 51, 71, 72, 73]. The idea behind these methodologies is to inject logical knowledge into the learning process through the loss function. The common strategy consists in translating logical rules into a regularization term to influence the training process: if the logical constraints are not satisfied during the optimization, then the regularization term will increase the loss.

  Starting from [72], Xu et al. propose a *semantic loss function* to maximize the probability that the provided logical knowledge is true. In this approach, logical formulas are expressed with propositional logic. Going on we find Semantic-Based Regularization (SBR) [71] by Diligenti et al., where the authors propose a multilayer architecture: a kernel machine is used as the base model and a neural network is used to approximate fuzzy predicate logic

  Another approach we have to mention Logic Tensor Networks (LTN) by Badreddine et al. [51]. LTN can be used in a wide range of machine learning applications. The authors propose the language of *Real Logic* to express FOL formulas. Predicates are represented as multiple neural networks and the final predictions are obtained by aggregation. LTN also includes in its features guarded quantifiers, explicit domain declarations, and different reasoning modalities. Moreover, it can be used as a generative model thanks to its ability to learn logical functions.

  Another interesting approach is proposed by Marra et al. with LYRICS [68]. Inspired by SBR and LTN, it defines a declarative language that is suitable for exploiting logical knowledge in any machine learning context. The same authors extended this work in [69] by introducing a novel class of loss functions. These functions are specified through a *t-norm generator* that leads to faster convergence. The difference can be found in the way the satisfaction of the knowledge is computed: instead of computing the truth degree of each subformula, connectives and quantifiers are used to combine the loss values in a less costly way.

  Hu et al. propose in [73] a different methodology to influence the optimization procedure. A distillation strategy is used to train the model: while the teacher network is trained in a semi-supervised setup to learn from unlabeled examples, the student network is trained to emulate the teacher and, at the same time, to predict some labeled examples.

The latest approach of this category is proposed by Fischer et al. with DL2 [70]. The model optimization is performed through an optimizer-adversary game: while the optimizer is trained to meet the constraints, the adversary is trained to find counterexamples that contradict the knowledge. Regarding the language, the knowledge is expressed through numerical comparisons linked by logical operators. This particular way of defining the knowledge makes this approach suitable to capture constraints in regression tasks.

- ***Logical knowledge applied to neural networks predictions:*** [74, 75, 76]. These approaches are used to compute logical inference starting from the output of a base neural network. The final output corresponds to the consequences implied by the neural predictions.

  In ABL [74] by Dai et al., the output of a base neural network is extended using abductive reasoning. The goal is to maximize the consistency between predictions and background knowledge. Given the background knowledge, the consistency optimization process is performed to revise the pseudo-labels produced by the network taking into account the hypotheses obtained through logical abduction.

  Li et al. [76] provide an alternative architecture to process and update neural network predictions. A grammar model is introduced to bridge the neural model and the symbolic reasoning module. Errors are propagated through a back-search algorithm to find the most probable corrections that will be used as pseudo-labels.

  The last approach we find is DeepProbLog [75], proposed by Manhaeve et al. as an extension of ProbLog. The goal of this work is to adapt ProbLog to deal with the output of a neural network. In this way, the predicates assume a neural interpretation. The logical knowledge is expressed through a function-free language in the form of a Prolog program, i.e., a set of Horn clauses.

Approaches of interest After this broad overview of possible NSI approaches, we can focus on those best suited to Fine-grained Entity Typing. In section 2.2, FET has been described as a multilabel multiclass problem where hierarchical dependencies occur between types. Given this premise, the approaches that explicitly support multilabel classification are LTN [51], SBR [71], LYRICS [68], KENN [8] and RNM [66]. During the categorization, we saw that LTN, LYRICS, and SBR, belong to the group of methods based on the loss function. On the contrary, KENN and RNM both have a post-elaboration step to modify the final predictions. A detailed comparison between loss-based approaches, KENN and RNM can be found in [9]. The more relevant differences will be highlighted below.

KENN is the most recent of the five approaches identified. As stated by its authors, the strategy used by KENN affects the Hypothesis Space (HS) differently from loss-based approaches: while the action of constraining the loss

function can be seen as "removing" solutions from the HS, the action of KENN will result in the opposite. In other words, LTN, LYRICS, and SBR will produce models where the solutions penalized at training time can no longer be obtained at inference time. On the other hand, the additional layer introduced by KENN keeps influencing the model at inference time by adding new solutions to the HS. Moving on to RNM, we can find a relevant difference with respect to KENN in the optimization of the clause. The methodology applied by RNM considers the entire logical knowledge, while KENN optimizes each clause separately. As we will see when presenting KENN in section 3, the independent optimization of the logical clauses may produce the side effect of not fully exploiting the knowledge during the training process.

All these works adopt a language that allows expressing FOL formulas. KENN is the only one having restrictions on the usage of connectives, quantifiers, predicates, and functions. However, these limitations still allow expressing the hierarchical relations of interest for FET. About the evaluation of a logical clause, each approach computes the truth degree of a formula using fuzzy operators. This feature makes it possible to build an end-to-end fully differentiable architecture except for RNM, which needs to solve an optimization problem at inference time.

An important feature to consider about an NSI system is the ability to learn clause weights. While KENN and RNM let you set clause weights as learnable parameters, LYRICS and SBR only support fixed weights. In LTN it is not possible to specify weights, but the authors suggest adding them indirectly to a clause by the conjunction of a 0-ary predicate. The feature of learning clause weights may be really helpful when dealing with multilabel classification problems because label dependencies are often unknown. In these cases, we can use a set of automatically generated rules and delegate to the network the choice of optimal weights. The learning process will decrease the weights of the clauses that are inconsistent with the data until they become irrelevant and, at the same time, it will increase the weights of the consistent ones.

The summary of the differences between the analyzed approaches can be found in Table 3.

Table 3: Comparison of KENN, RNM, LYRICS, LTN and SBR main characteristics

	KENN	**RNM**	**LYRICS**	**LTN**	**SBR**
Knowledge	FOL	FOL	FOL	FOL	FOL
Semantics	fuzzy	fuzzy	fuzzy	fuzzy	fuzzy
Restrictions / Assumptions	- only disjunction and negation - no parenthesis - universal assumption	none	none	none	none
Predicates	unary and binary	n-ary	n-ary	n-ary	n-ary
Functions	no	yes	yes	yes	yes
Clause weights	learnable	learnable	fixed	none	fixed

About the effectiveness of these works, in [66] we can find an experimen-

tal evaluation of collective classification to compare RNM and SBR. The results show that RNM achieved better performance than SBR on Citeseer [77] dataset. The same experiment is reported in [9] using KENN. In this case, the results show that KENN is less performing than RNM when dealing with small training sets. However, it becomes more effective when the percentage of training data increases. Another evaluation of KENN involves the datasets Yeast [78], Emotions [79] and VRD [80]. On these multilabel classification datasets, KENN reached better results than LTN and other state-of-the-art solutions in almost every setup [8].

Another important factor to take into account is scalability. In terms of computation, the way KENN injects the knowledge is quite simple and inexpensive with respect to the approaches that are based on the loss function. If we consider LTN as an example, expensive operations are performed at training time to compute the truth degree of a grounded formula. RNM has scalability issues too, but in this case the bottleneck can be found in the optimization problem to solve at inference time.

To sum up, KENN has more restrictions and limitations in the language. Anyhow, its expressivity suffices to represent the logical rules of interest. The advantages coming from learnable clause weights, scalability, and effectiveness, had a heavier impact on the choice of this NSI framework. Moreover, KENN is simple to integrate and analyze, thus allowing to easily study the effect of the logical knowledge injected by each clause.

3 Knowledge Enhanced Neural Networks

3.1 KENN

As anticipated in section 2.4.2, KENN is the Neuro-Symbolic Integration approach chosen for this study. It adopts a restricted FOL language with fuzzy semantics and injects the logical knowledge directly into the model structure by adding a special layer. This new layer acts to increase the satisfaction of the logical knowledge by modifying the final predictions of the neural network.

In this section we go deeper into the details of the framework of KENN by focusing on the most important aspects to better understand its role in this thesis.

3.1.1 Language and semantics

We can start by introducing more details about the logical language supported by KENN. This framework allows imposing constraints on predictions by providing a set of logical clauses that constitute our knowledge base. For the sake of brevity, we will refer to a logical knowledge base as KB. Each clause of the KB represents a logical formula generated from a subset of the FOL with the following restrictions:

- *the only operators allowed are disjunction and negation*
- *parentheses are not allowed*
- *variables are assumed to be universally quantified*
- *only unary and binary predicates are allowed*
- *functions are not allowed*

Even if we can use only the operators of negation and disjunction, it is possible to use logical equivalence to introduce some other operators into KENN's language. Considering classical logic for simplicity, two examples of logically equivalent formulas are:

- $A \rightarrow B = \neg A \vee B$
- $\neg(A \wedge B) = \neg A \vee \neg B$ (De Morgan's law)

The mapping between FOL and KENN's restricted language is quite trivial. If we consider the example of logical implication, we can schematize the translation as follows:

1. Starting from the FOL formula:

$$\forall X, A(X) \rightarrow B(X)$$

2. Remove quantifier (universal assumption):

$$A(X) \rightarrow B(X)$$

3. Remove variables[3]:

$$A \rightarrow B$$

4. Use logical equivalence (when possible) to convert unsupported operators:

$$\neg A \vee B$$

Now that we transformed the FOL formula into a clause supported by KENN, we have to convert it using the syntax required by the parser:

1. Use "n" as negation symbol and "," to represent disjunctions:

$$nA, B$$

2. Add a positive clause weight; use "_" in case you want to set it as learnable parameter:

$$0.5 : nA, B$$

The semantics of KENN's logic is fuzzy. The authors propose the *Gödel's t-conorm* as fuzzy operator to compute the truth degree of a formula. Given a clause c representing a disjunction of n literals, the value of the t-conorm is computed by:

$$\perp(c) = \max_{i=1}^{n}(c_i)$$

According to the equation, we can remark that the satisfaction of a clause depends exclusively on the value of the highest literal appearing in it.

3.1.2 Architecture

Moving on to KENN's architecture, we already saw in section 2.4.2 that it differs from most of its competitors that inject knowledge through the loss function. The idea behind this approach is to have an additional layer that is responsible for injecting knowledge. This layer is simply put on top of an arbitrary neural network and becomes an integral part of it, thus influencing both the learning process and the predictions at inference time according to the provided logical knowledge. The architecture of KENN is represented in Figure 5 at the maximum abstraction level.

[3]we can remove variables because we are dealing only with unary predicates referred to the same variable

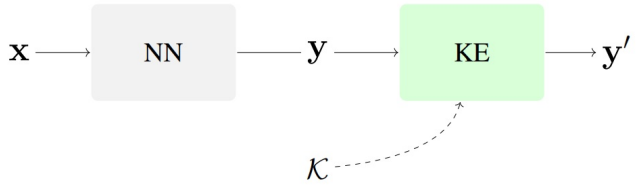


Figure 5: Architecture of KENN - Features x are given in input to a neural network (NN), the predictions y are modified by the Knowledge Enhancer (KE) to satisfy a logical KB (κ), thus obtaining the final predictions y'. (Source: [8])

Components
As we can see from the Figure 5, the main component of the architecture is the Knowledge Enhancer, which has a specific role and is composed of a set of subcomponents that cooperate:

- **Knowledge Enhancer (KE):** The KE is the layer responsible for injecting logical knowledge into the model. Given a logical KB K composed of n clauses, the KE contains n independent *Clause Enhancers* that produce n variations to modify y accordingly to K. These variations, also called deltas, are aggregated by the KE to obtain the enhanced output y'.
- **Clause Enhancer (CE):** The CE is the unit responsible for increasing the satisfaction of a clause by applying a *boost function*, which acts on the predictions of the labels related to the literals of the clause. Each CE is associated to a clause weight w_c that regulates the influence of the clause during the enhancement process.
- **T-conorm boost function (TBF):** The TBF is the function applied by each CE to increase the value of the Gödel t-conorm of its grounded clause. The TBF produces the deltas that will be aggregated by the KE to obtain the final predictions. Since the TBF is based on the Gödel t-conorm, which is non-differentiable, the authors propose the following soft differentiable approximation to compute the deltas:

$$\delta_{w_c}(v)_i = w_c \cdot softmax(v)_i \tag{1}$$

  where w_c is the weight of the clause c, v is the preactivation vector (i.e., the output of the base neural network) of the literals belonging to c, and i refers to the *i-th* literal of the clause.

Clause Enhancer details
Now that the high-level architecture has been presented, we can go deeper into the details of the clause enhancement mechanism. Referring to Figure 6, the functioning of a CE can be summarized with the following steps:

1. Receive the preactivations z of all the grounded literals (i.e., the output y of the base neural network).

2. Apply a pre-elaboration step ϕ to filter the preactivations: keep only the preactivations of the literals belonging to the clause c and change the sign of the preactivations of negated literals.

3. Apply the TBF (Equation 1) to the resulting preactivations to produce the deltas.

4. Apply a post-elaboration step ϕ' to convert the TBF's results into changes to be applied to the original preactivations z: expand back the dimensionality by filling with 0s the positions of the previously filtered out preactivations, then change back the sign of the deltas of negated literals.

An example of clause enhancement that may help to better comprehend the mechanism is shown in Figure 7.

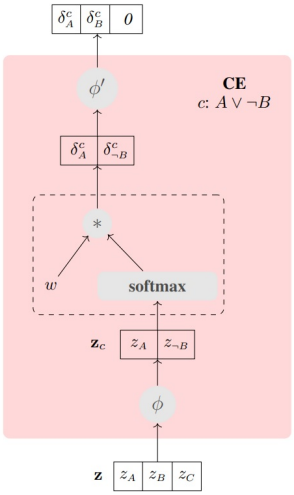


Figure 6: Clause Enhancer for $A \vee \neg B$. (Source: [8])

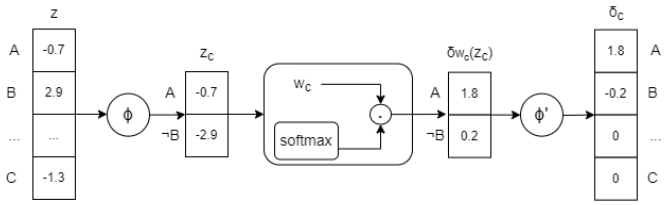


Figure 7: Example of clause enhancement for $A \vee \neg B$, with $w_c = 2.0$

Aggregation step
Once understood the enhancement mechanism for a single clause, it is necessary

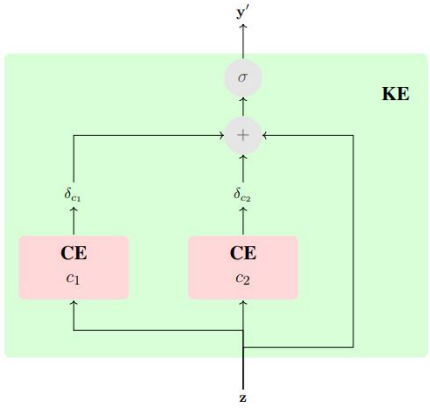


Figure 8: Knowledge Enhancer producing enhanced predictions. The deltas of each CE are summed to the preactivations of the base neural network. (Source: [8])

to explain how the outputs of each CE are combined. An illustration of this step is available in Figure 8. As we can see, the KE aggregates the initial preactivations to the deltas coming from each CE. The aggregation consists of a simple sum and the resulting value is passed through an activation function to obtain the final predictions y'. The authors of KENN motivate the choice of the sum as it leads to faster learning and inference, thus resulting in higher scalability. However, this kind of aggregation may lead to the so-called *conflicts*.

A conflict occurs when the same literal appears with different signs in two or more clauses, thus leading the CEs to produce for the same literal both positive and negative deltas. The natural consequence is that there could be side effects when performing the aggregation step since the benefits of a clause could be mitigated or completely overwhelmed by the effects of another clause. To better understand this particular situation, imagine having two clauses $c_1 : \neg A \vee B$ and $c_2 : \neg B \vee C \vee D$, where B is the literal involved in possible conflicts. Let us assume that starting from an arbitrary preactivation vector z the CEs generate $\delta(z_{c_1})_B = 0.7$ and $\delta(z_{c_2})_B = -0.5$. At this point, we can aggregate the deltas obtaining $\delta_B = 0.7 - 0.5 = 0.2$. The result is that the final change on B will reflect the effect of the stronger grounded clause lowered by the effect of the weaker one. It is important to underline that the more literals involved in a clause, the less the probability to encounter relevant conflicts, since the same literal must be dominant in both clauses [8].

Enhancement of a logical implication

An important aspect to remark and keep in mind is that since the goal of KENN is to increase the Gödel t-conorm of a grounded clause, the action of a CE is always the following: *produce a positive boost to the preactivations of positive literals and a negative boost to the preactivations of negative literals.* In

other words, the sign of a delta computed by the CE reflects the sign of the related literal of the clause. Thus, reminding that a logical implication must be expressed in KENN by using the logical equivalence $A \rightarrow B = \neg A \vee B$, the effect of a CE always results in a decrease of the preactivation of A and an increase of the preactivation of B. This means that a violated grounded clause (i.e., $1 \rightarrow 0$) is always pushed by KENN towards its satisfiability (i.e., $1 \rightarrow 1$, $0 \rightarrow 0$, or $0 \rightarrow 1$).

If we consider again the example in Figure 7, which is based on the clause $A \vee \neg B$ that can be seen as $B \rightarrow A$, we can analyze how the knowledge enhancement of an implication would modify the initial predictions. Table 4 reports the values obtained at each step of the CE, plus the prediction on A and B before and after the enhancement. If we consider $threshold = 0.5$ to assign 1 and 0 to the grounded literals and we compare the columns $\sigma(z)$ and $\sigma(\delta_{w_c}(z) + z)$, we can observe that KENN changed $1 \rightarrow 0$ into $1 \rightarrow 1$.

Table 4: Numeric example of the action of KENN for $A \vee \neg B$

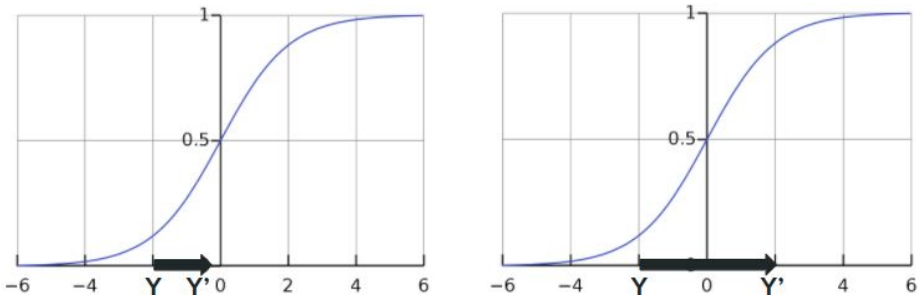
	z	$\sigma(z)$	z_c	$\delta_{w_c}(z_c)$	$\delta_{w_c}(z)$	$\delta_{w_c}(z) + z$	$\sigma(\delta_{w_c}(z) + z)$
A	-0.7	0.33 (≈ 0)	-0.7	1.8	1.8	1.1	0.75 (≈ 1)
B	2.9	0.95 (≈ 1)	-	-	-0.2	2.7	0.94 (≈ 1)
$\neg B$	-	-	-2.9	0.2	-	-	-

Considerations on the clause weight

In the example above, KENN with its intervention changed the final predictions leading to a satisfied clause. This is not always the case, since the choice of a different clause weight could lead to another outcome. In Figure 9 we can see two examples that start from the same preactivations, but use different clause weights. The examples show in a graphical way the changes introduced by KENN to the initial prediction from the perspective of the sigmoid activation function. Considering that a classification threshold of 0.5 on post-sigmoid values is equivalent to using a 0 threshold directly on preactivations, we can see that the delta produced in Figure 9a is not enough to push the prediction over the threshold. On the contrary, Figure 9b shows that the prediction changes its sign thanks to a higher clause weight.

Relational KENN

The authors of KENN propose also an extended version of the framework to use logical knowledge in relational domains. In that version, it is possible to specify relations between examples in contexts like collective classification by using binary predicates. The architecture of the relational version is slightly different and requires more sophisticated steps to inject logical knowledge into the neural network. The details can be found in [9] and will not be discussed here since this thesis focuses on the use of unary predicates.



(a) $w = 2$, $\delta_Y = 1.6$, $Y' = -0.4$ (b) $w = 5$, $\delta_Y = 4$, $Y' = 2$

Figure 9: Example of enhancement given $Y = -2$ and $softmax_Y = 0.8$, from the perspective of the sigmoid activation function

3.2 Preliminary experiments: encoding hierarchical rules

Before diving into the FET task, KENN has been tested in a hierarchical classification scenario. Since the goal of this preliminary phase is to familiarize with KENN rather than achieve the best performance, we do not dwell on the architecture, training setup, and results. We focus, instead, on how hierarchical relations can be encoded with KENN. Different heuristics to represent a hierarchy through logical rules are proposed within these preliminary studies.

3.2.1 Dataset

The dataset used for this experiment is called DBpedia Classes[4]. It is composed of approximately 240k labeled Wikipedia articles, where the only feature is the span of text. The types are organized as a forest of 9 trees, each of them representing a 3-level hierarchy. The type set counts 298 types: 9 at top-level, 70 at middle-level, and 219 at low-level. Each example is labeled with exactly 3 classes from the full path of a tree (i.e., one class per level). For example, if we consider the *Species* tree shown in Figure 10, an instance of the dataset could present $label_1 = Species$, $label_2 = Animal$, and $label_3 = Bird$.

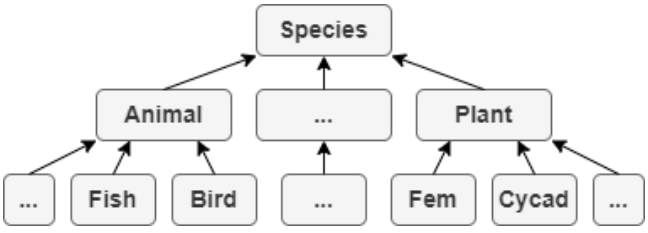


Figure 10: Example of a tree of DBpedia Classes

[4]https://www.kaggle.com/danofer/dbpedia-classes

3.2.2 Knowledge definition

The first step to integrate KENN is to create the logical KB. We can start from the tree in Figure 10 and then generalize different strategies for defining logical rules. The most intuitive and straightforward way consists in following the *is-a* relations, which are implicitly encoded in the tree structure. Considering the picture, it is possible to state that if an instance is a `Fish`, then it is an `Animal`. In the same way, we can then proceed with the next level of the hierarchy and state that if an instance is an `Animal`, then it is a `Species`. This mechanism can be extended to all the other subtrees to create a logical clause per edge. The resulting KB will be a set of logical implications between subtypes and supertypes. Even if it seems a natural way to describe the hierarchy, it doesn't necessarily mean it is the most effective solution. For this reason, we should take a step back and analyze the alternatives offered by the hierarchy. Figure 11 shows a generic hierarchy and highlights two groups of constraints we can distinguish:

1. **Vertical constraints:** based on the paths of a tree, they can be used to represent specialization relations
2. **Horizontal constraints:** based on pairs of nodes from different branches, they can be used to represent disjointness relations

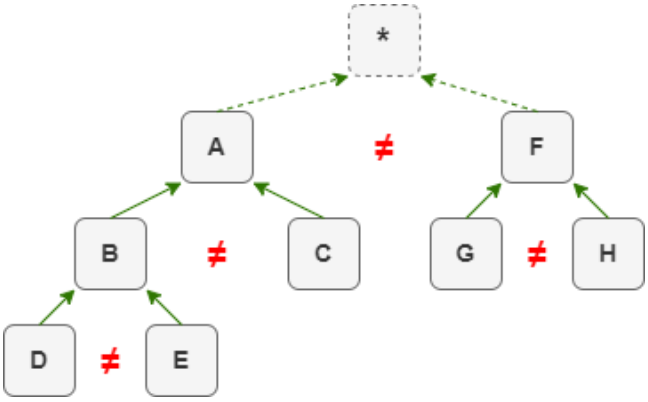


Figure 11: Examples of a generic hierarchy. In green the vertical constraints, in red the horizontal constraints.

For each group of constraints, it is possible to implement different strategies to define the KB. In the rest of the thesis, these strategies will be referred to as *KB modes*. Note that in the definitions below we will use unary predicates to represent the belonging of an instance X to a type.

Vertical Constraints
Using the unary predicates *Subtype* and *Supertype* such that the type *Subtype*

is a specialization of the type *Supertype*, we can define three main strategies to build vertical constraints:

- **Bottom Up:**
  - FOL: $\forall X, Subtype(X) \rightarrow Supertype(X)$
  - KENN: $\neg Subtype \vee Supertype$
  - Defined by following every path of the tree from the leaves to the root. It represents the logic dependency between types, following the most widely used semantics in the definition of subclass.
  - Open World Assumption: a clause is satisfied even when an instance is assigned only to supertypes
- **Top Down:**
  - FOL: $\forall X, Supertype(X) \rightarrow (Subtype_1(X) \vee ... \vee Subtype_n(X))$
  - KENN: $\neg Supertype \vee Subtype_1 \vee ... \vee Subtype_n$
  - Defined by following every path of the tree from the root to the leaves
  - Closed World Assumption: a clause is not satisfied when an instance is assigned to a supertype without any of its subtypes
- **Hybrid:**
  - Composed of a mix of *Bottom Up* and *Top Down* clauses

Note that if a model that uses this kind of knowledge is confident about a given type, a *Bottom Up* clause propagates this certainty towards the root of the hierarchy. Conversely, a *Top Down* clause only guarantees that one of the subtypes is appropriate, but provides no information as to which one. The reason, of course, is the tree-like hierarchy, where each type has only one parent, but potentially many descendants.

The presented KB modes can be considered as the starting point to create new strategies by adding some variations. In Table 5 we can find a summary that comprehends every KB mode, each of them accompanied by examples of clauses from the hierarchy in Figure 10. Before proceeding with the explanation of the variants, we have to make clear why they have been introduced: the conflicts. If we look at the table, we can see that in the examples of the pure *Bottom Up* and *Top Down* modes there are conflicts: a middle-level label appears both as positive and negative literal. This fact derives from the translation of the logical implication into a disjunction since an intermediate node is at the same time antecedent and consequent of different logical implications. This behavior still hold for the *Hybrid* mode.

Now that the possible problems of the starting strategies are known, it is possible to introduce their conflict-free variants. These variants avoid conflict situations in different ways. The "Skip" ones rely on the transitivity of a relation by linking top-level and low-level nodes by skipping those middle-level nodes

that would create a conflict (i.e., a middle-level node cannot be the antecedent of an implication rule). These solutions provide a more shallow representation of the knowledge, especially for the *Top Down Skip*. In this case, a logical rule that represents the relation between top-level and low-level lose effectiveness because the implication may have too many consequents. The *Hybrid In* and *Hybrid Out* modes, instead, avoid conflicts by fixing the middle-level types as consequent or antecedent of an implication rule, respectively. In this way, the deltas produced by each CE for the same literal will always have the same sign.

Note that the proposed variants are valid for this dataset, but some of them do not apply to every context since they require a hierarchy with at least 3 levels.

Horizontal Constraints

These constraints aim to avoid the co-occurrence of types whose instances form disjoint sets (e.g., if an example is labeled as `Person` it cannot be labeled as `Location` and vice versa). Using T_i to indicate an arbitrary type, the logical rule that best represents this constraint is:

$$\forall X, T_1(X) \rightarrow (\neg T_2(X) \wedge ... \wedge \neg T_n(X))$$

With the following steps, we can derive an equivalent formula without implication and conjunctions:

$$\begin{aligned} & T_1 \rightarrow (\neg T_2 \wedge ... \wedge \neg T_n) = \\ & = \neg T_1 \vee (\neg T_2 \wedge ... \wedge \neg T_n) = \\ & = \neg T_1 \vee \neg (T_2 \vee ... \vee T_n) \end{aligned}$$

Unfortunately, the derived formula cannot be furthermore decomposed due to the presence of parentheses. We can opt for two alternative solutions to bypass the problem:

1. **Approximate the rule with a softer constraint:** if an instance does not belong to $\texttt{T}_1$, then it should belong to another type.
   The logical formula for this statement is:

   $$\forall X, \neg T_1(X) \rightarrow (T_2(X) \vee ... \vee T_n(X))$$

   that can be translated into KENN's language with the following steps:

   $$\begin{aligned} & \neg T_1 \rightarrow (T_2 \vee ... \vee T_n) = \\ & = T_1 \vee (T_2 \vee ... \vee T_n) = \\ & = T_1 \vee T_2 \vee ... \vee T_n \end{aligned}$$

   From a logical point of view, the resulting formula does not seem very helpful because it simply states that an instance belongs at least to one class. Indeed, if we repeat the previous steps using any other type as the

Table 5: Strategies for defining logical clauses based on the hierarchy in Figure 10

	Clauses
Bottom Up	$c_1 : \neg Fish \vee Animal$ $c_2 : \neg Bird \vee Animal$ $c_3 : \neg Animal \vee Species$ ...
Top Down	$c_1 : \neg Species \vee Animal \vee ... \vee Plant$ $c_2 : \neg Animal \vee Fish \vee ... \vee Bird$ $c_3 : \neg Plant \vee Fem \vee ... \vee Cycad$ ...
Hybrid	$c_1 : \neg Species \vee Animal \vee ... \vee Plant$ $c_2 : \neg Animal \vee Species$ $c_3 : \neg Plant \vee Species$ ...
Bottom Up Skip	$c_1 : \neg Fish \vee Animal$ $c_2 : \neg Fish \vee Species$ $c_3 : \neg Bird \vee Animal$ $c_4 : \neg Bird \vee Species$ ...
Top Down Skip	$c_1 : \neg Species \vee Animal \vee ... \vee Plant$ $c_2 : \neg Species \vee Fish \vee ... \vee Bird \vee Fem \vee Cycad$ ...
Hybrid In	$c_1 : \neg Species \vee Animal \vee ... \vee Plant$ $c_2 : \neg Fish \vee Animal$ $c_3 : \neg Bird \vee Animal$ $c_4 : \neg Fem \vee Plant$ ...
Hybrid Out	$c_1 : \neg Animal \vee Species$ $c_2 : \neg Animal \vee Fish \vee ... \vee Bird$ $c_3 : \neg Plant \vee Species$ $c_4 : \neg Plant \vee Fem \vee ... \vee Cycad$ ...

antecedent of the implication, the derived clause will be identical. Furthermore, the action of KENN on such clauses would not lead to mutual exclusivity because every literal is positive and none of the involved predictions will receive a negative boost.

2. **Split the rule into multiple clauses:** define a clause for each pair of disjoint types. Given two disjoint types, the logical formula to express the mutual exclusivity is:

$$\forall X, T_1(X) \rightarrow \neg T_2(X)$$

If we consider n disjoint pairs in which appears T_1, the resulting KB in KENN's language will be the following:

$$c_1 : \neg T_1 \vee \neg T_2$$
$$...$$
$$c_n : \neg T_1 \vee \neg T_n$$

Unlike strategy 1, this one preserves the soundness of the starting formula. However, there are still some issues to take into account. The first concerns the number of clauses necessary to cover the possible combinations of pairs: having n mutual independent types will result in $\frac{n!}{2!(n-2)!}$ logical clauses. The second issue regards the action of KENN: while in strategy 1 there were only positive literals, in this strategy there are only negative literals, thus meaning that none of the predictions will ever receive a positive boost.

Horizontal constraints - Alternative usage

Another use of horizontal constraints that is not related to this dataset, but could be useful in other contexts, is to encourage the co-occurrence between different types. Here the purpose is exactly the opposite with respect to the previous rules. Even if this goal can be reached by applying a simple inference on the final predictions without the use of KENN, in some tasks (e.g., domain adaptation) it may be useful to leave the decision to the network. If we want the model to learn a mapping between two types `A` and `B`, we can define the logical rule

$$(A \rightarrow B) \wedge (B \rightarrow A)$$

that can be expressed in KENN using the following clauses:

$$c1 : \neg A \vee B$$
$$c2 : \neg B \vee A$$

The first thing we can notice is the presence of conflicts, but they will not have a negative impact on the final predictions. The preactivation of the literals belonging to the two clauses are the same with opposite signs, so a literal cannot be dominant in both clauses. In other words, the final predictions always benefit from the knowledge enhancement since the effect of KENN will be one of the following:

- **positive preactivations:** a positive aggregated delta is produced for each literal
- **negative preactivations:** a negative aggregated delta is produced for each literal
- **discord preactivations:** an aggregated delta with the sign of the highest[5] preactivation is produced for each literal

3.2.3 Results

As anticipated, the results are not the focus of this study since we are not dealing with the target problem of this thesis. To see KENN in action, the KB modes of Bottom Up, Top Down, Hybrid In, and Hybrid Out have been evaluated. The baseline model implemented for these experiments has a simple architecture. It uses DistilBERT as encoder, followed by a dense fully connected layer and a final classification layer that applies the sigmoid activation function. The choice of this activation function is due to the fact that in a multilabel task we want as output the probability of each label. Starting from this architecture, KENN is placed between the fully connected layer and the classification layer. Clause weights are set as learnable parameters with an initial value of 0.5.

In Table 6 are reported the results obtained by the baseline and KENN-based models. Hybrid Out is the configuration that brought more benefits to the baseline model with a +0.0157 on the F1 score, thanks to its highest recall. Even Top Down brought improvements with an increase of +0.0097. This did not happen with the other configurations.

An interesting fact that emerged regards the clause weights. By inspecting the final weights learned by the models, it was found out that the models that performed the best are the ones that finished the train with higher clause weights. In particular, some clauses have increased their starting weight by 10 times. On the contrary, the worst models are the ones whose final weights are smaller. This fact means that KENN helped the baseline model the most when it gained more influence on the final predictions. From these results, we can say that the higher the weight, the better the performance. This result may lead to think that it could be reasonable to set higher initial clause weights to anticipate the learning process. However, we have to take this conclusion with a grain of salt because it could be strictly related to this dataset.

[5] in absolute value

Table 6: Comparison between the baseline and KENN-based models on DBpedia Classes in terms of *macro f1 classes*

KB mode	**P**	**R**	**F1**
- (baseline)	0.9265	0.8644	0.8897
Bottom Up	0.924	0.8685	0.8901
Top Down	**0.93**	0.8788	0.8994
Hybrid In	0.9186	0.87	0.8884
Hybrid Out	0.927	**0.8913**	**0.9054**

4 Fine-grained Entity Typing with KENN

We saw in section 2.2 that Fine-grained Entity Typing (FET) is a multilabel classification problem, where the goal is to assign one or more types from a type set to a mention appearing in a text. This section will present the FET datasets used for the experiments and the architecture of the adopted models.

4.1 Datasets

Several datasets were proposed through the years to evaluate the task of FET. Since building large annotated datasets is an expensive task, most of them are built using Distant Supervision techniques. Some state-of-the-art datasets used to evaluate FET approaches are FIGER [4], BBN [28], OntoNotes [6] and Choi [23]. While the first three datasets have tree-like structures to group types hierarchically, Choi is organized differently. It groups the types into three categories: general, fine-grained, and ultra-fine-grained. Each dataset is provided with its own type set and is split into training set and test set. While every training set is obtained through distant supervision, test sets are manually annotated for most of them.

This thesis focuses the experiments on two of the most popular datasets: FIGER and BBN. The reason behind the choice is that they have a less complex hierarchy than OntoNotes, thus allowing us to better comprehend how KENN influences the task and how to take advantage of it. Regarding Choi, it has not been considered for this work because types are not hierarchically organized.

4.1.1 FIGER

FIGER is one of the first FET datasets. It was presented in 2012 with Fine-Grained Entity Recognition [4]. The original version of the dataset is based on a type set T derived from Freebase [81] and counts a total of 112 types. The hierarchy is organized as a forest of trees with maximum depth 1, i.e., 2-level hierarchy. A summary of the type set can be found in Figure 12.

As said previously, the training set is obtained with a distant supervision technique and counts about $1.5M$ instances. The source of data that has been labeled is constituted by Wikipedia articles, where anchor links are treated as entity mentions. The annotation procedure can be summarized in three steps:

1. For each sentence, detect all the linked segments m
2. For each m, retrieve the related Wikipedia entity e_m
3. For each e_m, obtain its types t'_m from Freebase
4. For each t'_m, refine the types to obtain a subset $t_m \subseteq t'_m$ such that $t_m \subseteq T$. The refinement is done by keeping only the ones with more than 5 ground instances and merging types that are too specific.

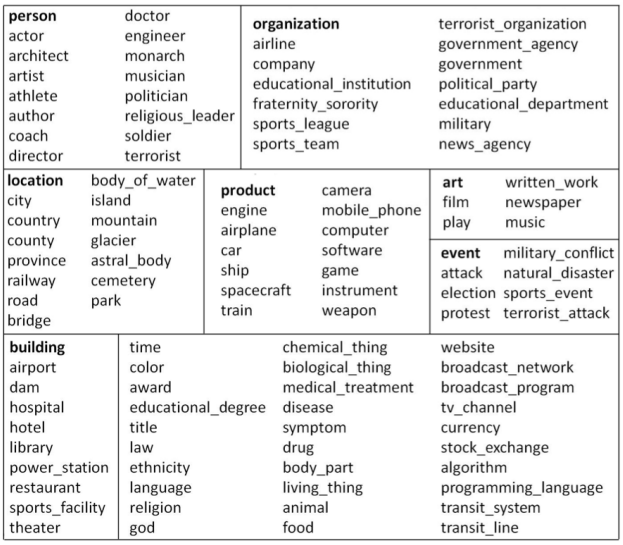


Figure 12: Type set of FIGER. Each box can be seen as a two-level tree, with the root node represented by the type in bold. The box at the bottom right is composed of uncategorized types. (Source: [4])


```
{
    "tokens": ["W.R.", "Grace", "holds", "three", "of", "Grace",
               "Energy", "'s", "seven", "board", "seats", "."],
    "mentions": [
        {"start": 0, "labels": ["/ORGANIZATION/CORPORATION", "/ORGANIZATION"], "end": 2},
        {"start": 5, "labels": ["/ORGANIZATION/CORPORATION", "/ORGANIZATION"], "end": 7}],
    "senid": 2, "fileid": "WSJ0005"
}
```


Figure 13: Structure of a FIGER/BBN entry. Example taken from BBN.

The test set is composed of 434 manually annotated examples. The sentences are extracted from local newspapers, photography and veterinary magazines, and the student newspaper of the University of Washington.

Each entry of the dataset is represented following the structure of the JSON in Figure 13. The main fields are *tokens*, which contains the list of the worlds of the sentence, and *mentions*. For each mention there are three subfields: *start* and *end* indicate the index of the tokens that delimits the mention, and *labels* contains the true types.

4.1.2 BBN

BBN dataset for FET made its first appearance in [29] in 2016. The original version of this dataset comes from the BBN Pronoun Coreference and Entity Type Corpus [5], composed of manually annotated Wall Street Journal articles. These articles were originally annotated using 93 types, divided into 12 named

```
{
    "left_context_token": ["According", "to", "analysts", ",", "profits", "were",
         "also", "helped", "by", "successful", "cost-cutting", "measures", "at"],
    "mention_span": "Newsweek",
    "right_context_token": ["."],
    "y_str": ["/ORGANIZATION/CORPORATION", "/WORK_OF_ART",
              "/ORGANIZATION", "/WORK_OF_ART/BOOK"]
}
```


Figure 14: Structure of a FIGER/BBN entry after data preparation - Example taken from BBN

entity types, 9 nominal entity types, 7 numeric types, and 64 subtypes. However, when applying distant supervision, the authors used DBpedia Spotlight to link entities and they had to remove those BBN types that could not be mapped into Freebase types. The resulting type set counts 47 types.

The dataset is constituted by 32,739 entries. The test set has not been manually annotated. To construct it, the authors simply extracted a subset composed of the 20% of the dataset,

An entry of the dataset is structured in the same way as FIGER (Figure 13), so we can find the sentence as *tokens* and the *mentions* including *start*, *end* and *labels* as subfields.

4.1.3 Dataset preparation & statistics

The adopted versions of the datasets are those proposed by Ren et al. in [28][6]. Before performing the experiments, both training sets of FIGER and BBN were refined. First of all, the duplicates (i.e., instances with the same context, mention, and types) were removed. Then, examples having the same context and different sets of mentions were merged. Each entry with more than one mention was subsequently divided into distinct entries with single mentions. Regarding FIGER, the equivalent types `/living_thing` and `/livingthing` were unified in `/living_thing`. Finally the sentences were split into 3 fields: *left_context_token*, *mention_span* and *right_context_token*. The mention types can be found in the field *y_str*. An example of an entry of the refined dataset is available in Figure 14. Since the dev sets were not provided within the original datasets, they were obtained by sampling about 1K instances of the training sets, such that single-mention examples derived from the same multi-mention example are assigned to the same set.

Before proceeding with the analysis of the datasets, it is helpful to point out that every sentence can contain multiple types from different branches. We will refer to this condition with the term multibranch. Furthermore, if an instance x of the dataset is labeled with a type t that is a subtype of t', then the type set of x must contain both t and t'.

In Table 7 we can find some useful statistics about the two datasets. The fields of the table can be described as follows:

[6] https://github.com/INK-USC/AFET

- **# ex:** number of examples of the refined version of the dataset
- **% only supertype:** percentage of examples where a coarse type occurs without any of its subtypes
- **% only supertype (strict):** percentage of examples where only coarse types occur
- **% inter-tree:** percentage of examples where sibling types occur together
- **% intra-tree:** percentage of examples where types from different trees occur together
- **% intra/inter-tree:** percentage of examples where types from different branches occur together[7]
- **# types:** number of types of the refined version of the dataset
- **# avg types:** average number of types per example
- **% isolated types:** percentage of types corresponding to an isolated node[8] of the hierarchy

Looking at the table, it is possible to draw some considerations. In both datasets, we can observe that the multibranch examples are mostly caused by the presence of *inter-tree* types. Regarding FIGER, we see that the statistics about *only supertype* are widely different between the training/dev and test sets. We have about the 21% of cases against the 60%, becoming 15% against 54% considering the *strict* mode. This diversity derives from the manual annotation, where probably the types annotated through distant supervision that were too specific have been removed by the annotators. Even the statistics of *intra/inter-tree* differ a lot between the training/dev and test sets. In this case, we move from the 35% of multibranch examples to the 11%. Again, the reason can be found in the manual annotation. Indeed, when using distant supervision techniques, a mention can be labeled with all its meanings disregarding the context. To point out how noisy is FIGER, we can also see how the *# avg types* statistic drastically decreases after the manual annotation.

Moving on to BBN, which does not have a manually annotated test set, the situation is quite different. First of all, we can notice that while the training/dev set counts the 22-24% of multibranch examples, in the test set they are absent. This statistic is motivated by the fact that Ren et al. pruned these examples from the test set. For the same reason, we have no difference between the values of *% only supertype* and *% only supertype (strict)* in the test.

The information about the hierarchy of each dataset are reported in Table 8. The columns of the table can be described as follows:

[7] this statistic is not equivalent to *% inter-tree* + % intra-tree, since in the same example we could find both the situations

[8] node without any parent or child

Table 7: Datasets statistics

Dataset	# ex	% only supertype	% only supertype (strict)	% inter-tree	% intra-tree	% inter/intra-tree
BBN train	84,492	16.0	7.64	22.33	2.65	24.09
BBN dev	1,039	15.3	8.47	20.69	2.31	22.52
BBN test	12,349	7.44	7.44	-	-	-
FIGER train	2,676,854	21.33	14.99	23.63	15.38	35.23
FIGER dev	1,094	21.57	14.72	25.59	14.35	35.83
FIGER test	563	60.57	54.17	11.01	0.71	11.72

Dataset	# types	# avg types	% isolated types
BBN train	157,292	1.86	24.3
BBN dev	1,903	1.83	24.44
BBN test	20,606	1.67	15.4
FIGER train	6,380,532	2.38	9.21
FIGER dev	2,616	2.39	9.79
FIGER test	840	1.49	6.55

Table 8: Datasets hierarchical structure

Dataset	# nodes	# levels	# top level	# leaves	# isolated
BBN	47	2	9	31	7
FIGER	128	2	22	79	27

- **# nodes:** number of nodes of the hierarchy (i.e., cardinality of the type set)
- **# levels:** maximum number of levels of a tree in the type set
- **# top level:** number of root nodes with at least a child (i.e., supertypes)
- **# leaves:** number of child nodes (i.e., subtypes)
- **# isolated:** number of isolated nodes (i.e., neither supertypes nor subtypes)

4.2 Baseline model

The architecture of the baseline network used in the experiments is shown in Figure 15. Starting from the input, the text is composed of three parts: left context (LC), mention (M), and right context (RC). The encoder receives a sequence in the form of `[CLS]M[SEP]LC[SEP]RC`. The example in the picture reports BERT as encoder, but any variant could be used to take its place. The versions used to perform the experiments of this thesis are BERT large[9] and DistilBERT[10] for PyTorch[11]. In any case, the encoders are trained using the adapters with the Pfeiffer strategy. The framework used to integrate them into the models is AdapterHub[12]. Once the encoder produces the output sequence, the encoding of the `[CLS]` token is fed into a basic neural network composed

[9] https://huggingface.co/bert-large-cased
[10] https://huggingface.co/distilbert-base-uncased
[11] https://pytorch.org/
[12] https://github.com/Adapter-Hub/adapter-transformers

of a fully connected layer and a classification layer. The fully connected layer has the same dimension as the encoder's output, the classification layer applies the sigmoid activation function to compute the probability to assign each label. Finally, there is the inference step to obtain the final predictions starting from the probabilities (more details in section 4.4).

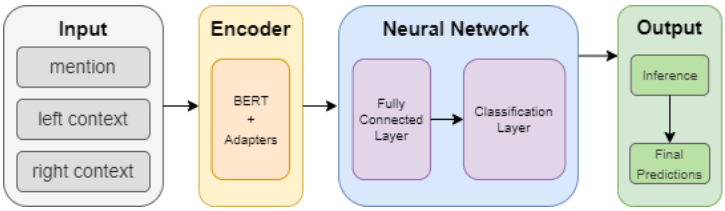


Figure 15: Baseline model architecture

4.3 KENN model

The architecture of the KENN-based model is designed starting from the baseline. As we can see in Figure 16, the architecture of the two models is almost the same. The only differences are the presence of KENN in the middle of the network and the KB provided to the model. The knowledge enhancement layer is placed between the fully connected layer and the classification layer. What said for the baseline model about input representation, encoders, etc. is still valid for this model.

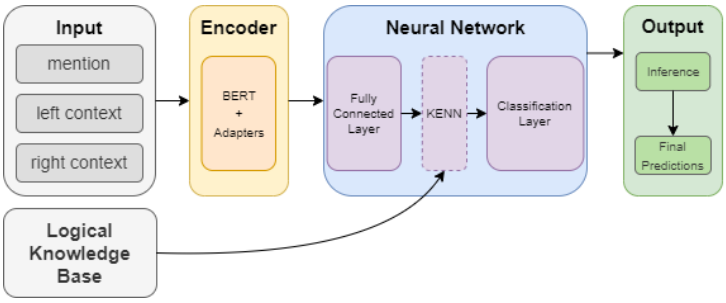


Figure 16: KENN-based model architecture

Among the strategies for defining logical KB proposed in section 3.2.2, the ones chosen for the experiments are the *Bottom Up*, *Top Down*, and *Hybrid*. Other KB modes from Table 5 are not taken into account since they do not apply to datasets with a 2-level hierarchy. We will not even analyze horizontal constraints since the priority has been given to the vertical ones. The number of clauses generated by the chosen KB modes are reported in Table 9.

Depending on the KB mode, KENN affects the predictions of the baseline network in different ways. Some statistics about the influence of the logical knowledge are available in Table 10, which provides the following information:

Table 9: Datasets and number of clauses

Dataset	Bottom Up	Top Down	Hybrid
BBN	31	9	40
FIGER	79	22	101

Table 10: Datasets statistics on KENN's influence

Dataset	% no clauses	% no clauses (strict)	% inappropriate Top Down rules	% inappropriate Top Down rules (strict)
BBN train	44.74	37.01	16.00	7.64
BBN dev	44.47	37.05	15.30	8.47
BBN test	25.69	25.69	7.44	7.44
FIGER train	18.06	9.85	21.33	14.99
FIGER dev	20.02	10.79	21.57	14.72
FIGER test	9.59	5.33	60.57	54.17

- **% no clauses:** percentage of examples labeled with at least one type that does not occur in any clause (i.e. isolated type)
- **% no clauses (strict):** percentage of examples labeled exclusively with types that do not occur in any clause (i.e. isolated types only)
- **% inappropriate Top Down rules:** percentage of examples in which a supertype occurs without any of its subtypes; equivalent to *% only supertype* in Table 7
- **% inappropriate Top Down rules (strict):** percentage of examples in which only supertypes occur; equivalent to *% only supertype (strict)* in Table 7

With the term *"inappropriate"* used in the last two statistics, we mean that the intervention of KENN may be undesired. Contrarily to what happens in Bottom Up, the knowledge enhancement could be disadvantageous when using the Top Down mode in some cases. The reason is that a Top Down clause will always encourage the model to predict subtypes even when the ground truth contains only supertypes. Indeed, we saw in section 3.2.2 that while Bottom Up respects the Open World Assumption, Top Down is defined under the Closed World Assumption. About the statistics from the table, we can notice relevant differences between the train/dev and the test sets. The most evident one regards FIGER and the *Top Down* statistics, where we can observe a difference of 40 percentage points.

4.4 Inference rules

The scores produced by the model can be processed through different inference rules to transform the probability of each class into a binary prediction. The

most common strategy to discriminate between positive and negative predictions in classification problems is to use a threshold, but there are many alternatives that may lead to better results. The choice of the inference rule may heavily affect the evaluation of a model. Given an instance, we indicate with Y the set of output values of the neural network and with y the output for a specific type. The inference rules that will be used to compute the final predictions are the following:

1. **Threshold:** basic inference method which simply compare the scores with the threshold

$$pred(y) = 1 \iff score(y) > threshold$$

2. **Threshold or max:** same as *threshold*'s method, but it prevents void predictions by assigning the class with maximum score when none of the scores exceeds the threshold

$$pred(y) = 1 \iff (score(y) > threshold \vee score(y) = \max_{y_i \in Y} score(y_i))$$

   This inference rule is commonly used by FET from the literature.

5 Experiments

The conducted experiments involve many parameters that can be varied and combined to observe KENN's behavior from several points of view. The space of parameters is the following:

- **KB mode:** the KB encoding strategy used to define the logical knowledge
- **initial clause weight:** the value used to initialize the network parameters of the clause weights
- **fixed/learnable clause weights:** the strategy used to set the clause weights
- **encoder:** the language model used to encode the textual input
- **loss function:** the loss function used during training

Before describing the experiments, we will introduce in the next sections the ET metrics and the training setups adopted. It is also important to point out that every model used the same initialization of random seeds, which guarantees:

1. Same layer initialization between different setups
2. Given an epoch, each model is trained on the same batch of data

5.1 Metrics for Entity Typing

The evaluation of an ET approach relies on some variants of the metrics used in classical classification problems. The explanation of each metric will be supported by numerical values obtained from the example in Table 11.

Table 11: Example of predictions over a type set $T = \{T1, T2, T3\}$ and a dataset $X = \{X1, X2, X3\}$

	Prediction			**Target**		
Example	**T1**	**T2**	**T3**	**T1**	**T2**	**T3**
X_1	x		x	x	x	
X_2		x		x		x
X_3	x		x	x		x

In the definition of the metrics we will use terminology that needs a preliminary explanation:

- True Positive (**TP**): number of correct positive predictions. In the example in Table 11 we have TP=3 .
- False Positive (**FP**): number of wrong positive predictions. In the example in Table 11 we have FP=2 .

- True Negative (**TN**): number of correct negative predictions. In the example in Table 11 we have TN=1 .
- False Negative (**FN**): number of wrong negative predictions. In the example in Table 11 we have FN=3 .

Once fixed these preliminary notions, the evaluated metrics are defined as follows:

- **Accuracy:** percentage of correct predictions; defined as

$$\frac{count(Prediction == Target)}{|X|} = \frac{1}{3}$$

- **Micro precision:** percentage of correct positive predictions with respect to all the positive predictions; defined as

$$\frac{TP}{TP + FP} = \frac{3}{3+2} = 0.6$$

- **Micro recall:** percentage of correct positive predictions with respect to the positive target; defined as

$$\frac{TP}{TP + FN} = \frac{3}{3+3} = 0.5$$

- **Micro f1:** harmonic mean of micro precision and micro recall; defined as

$$2 \cdot \frac{MicroPrecision \cdot MicroRecall}{MicroPrecision + MicroRecall} = 2 \cdot \frac{0.6 \cdot 0.5}{0.6 + 0.5} = 0.54$$

- **Macro precision examples:** mean of the percentages of correct positive predictions with respect to all the positive predictions of each example; defined as

$$\frac{\sum_{x \in X} \frac{TP(x)}{TP(x)+FP(x)}}{|X|} = \frac{0.5 + 0 + 1}{3} = 0.5$$

- **Macro recall examples:** mean of the percentages of correct positive predictions with respect to the positive target of each example; defined as

$$\frac{\sum_{x \in X} \frac{TP(x)}{TP(x)+FN(x)}}{|X|} = \frac{0.5 + 0 + 1}{3} = 0.5$$

- **Macro f1 examples:** harmonic mean of micro precision examples and micro recall examples; defined as

$$2 \cdot \frac{MacroPrecisionExamples \cdot MacroRecallExamples}{MacroPrecisionExamples + MacroRecallExamples} =$$
$$= 2 \cdot \frac{0.5 \cdot 0.5}{0.5 + 0.5} = 0.5$$

- **Macro precision classes:** mean of the percentages of correct positive predictions with respect to all the positive predictions of each class; defined as

$$\frac{\sum_{t \in T} \frac{TP(t)}{TP(t)+FP(t)}}{|T|} = \frac{1+0+0.5}{3} = 0.5$$

- **Macro recall classes:** mean of the percentages of correct positive predictions with respect to the positive target of each class; defined as

$$\frac{\sum_{t \in T} \frac{TP(t)}{TP(t)+FN(t)}}{|T|} = \frac{0.66+0+0.5}{3} = 0.38$$

- **Macro f1 classes:** harmonic mean of macro precision classes and macro recall classes; defined as

$$2 \cdot \frac{MacroPrecisionClasses \cdot MacroRecallClasses}{MacroPrecisionClasses + MacroRecallClasses} = \\ = 2 \cdot \frac{0.5 \cdot 0.38}{0.5+0.38} = 0.43$$

5.2 Experimental setups

Two setups were adopted for the experiments. The main differences between the setups are the stopping criterion and the batch size. In particular, *Setup B* is used when BERT is involved since *Setup A* would be too expensive.

Setup A

- **number of epochs:** 100 for FIGER, 75 for BBN[13]
- **number of examples per epoch:** 10,240 (20 batches[14] of size 512)
- **data shuffle:** each time the whole dataset has been seen
- **optimizer:** Adam with fixed learning rate set to 0.0005
- **inference:** threshold = 0.5

Setup B

- **number of epochs:** undefined; determined by an early stopping strategy on the *dev loss* with *patience = 5*
- **number of examples per epoch:** 10,240 (160 batches of size 64)
- **data shuffle:** every time the whole dataset has been seen
- **optimizer:** Adam with fixed learning rate set to 0.0005
- **inference:** threshold = 0.5

[13] BBN is smaller than FIGER and converges faster

[14] using n batches per epoch means performing n backward operations per epoch

5.3 Baseline hyperparameter search

The architecture of the baseline neural network was kept simple to have a basic starting point, so it did not involve any search of the best combination of hidden layers/units. The optimization aimed to find the best representation of the input text in terms of tokens and involved a few hyperparameters.

Fixed the training configuration following the *Setup A*, we can proceed with the explanation of the hyperparameter search. We saw in section 4.2 that the input of the model is split in mention, left context and right context tokens separated by `[SEP]`. Although keeping every token of every span would feed the model with the maximum amount of information, this is not a feasible solution because of the high cost. For this reason, it is necessary to find the optimal number of tokens for both mention and context to reach a trade-off between cost and performance. Before going on, an important remark: tokens and words are two distinct things. In this case, we are looking for the optimal number of words that will be transformed by the tokenizer into a different number of tokens. Since BERT's tokenization is based on the WordPiece, the number of tokens grows up rapidly due to the subwords segmentation. For this reason, the maximum number of tokens is capped to 80 (left-to-right) for not exceeding the resource usage. Sequences with less than 80 tokens are filled with empty tokens.

The optimization is based on the metrics of *macro f1 examples* and *macro f1 classes* in this order of relevance. The search is performed iteratively by changing the number of words to keep for each input part. Starting from 1 mention word without any context, the number of mention words is increased until it stops improving. Once fixed the length of the mention, the left and right contexts get increased until reaching convergence. The two contexts share the same length.

The best configurations found for the two datasets are the following:

- **FIGER:** `mention = 6`, `context = 19`
- **BBN:** `mention = 5`, `context = 13`

Since these parameters refer to DistilBERT, the search procedure should be repeated for BERT to use it at its best. However, this has not been done for cost reasons. Aware of this, the same parameters have been used for both the encoders.

5.4 Initial experiments with KENN

The goal of these experiments is to analyze how KENN behaves when involved in a FET task. When the framework has been introduced in section 3, we saw that it allows using different settings about clause weights (i.e., fixed values, learnable parameters). However, the best choice for this context needs to be discovered. The same goes for the KB modes, as several alternatives were proposed without knowing which one would be the most promising for our task. Given these

premises, the analysis can be divided into two parts. The first part is constituted by some *quantitative analysis* (section 5.4.3) and aims to study the impact of each configuration of KENN. The second part is called *preactivations analysis* (section 5.4.4) and investigates what happens in the network at a lower level.

5.4.1 Setup

Both the baseline and KENN-based models used in these experiments follow the *Setup A*. The tested configurations are obtained with the following parameters:

- **KB modes:** Bottom Up, Top Down and Hybrid
- **initial clause weights:** 0.5, 1.0 and 2.0
- **fixed clause weights**
- **encoder:** DistilBERT with adapters
- **loss function:** Binary Cross-Entropy, with the weights of positive examples set to 1

Each KB mode is tested by varying the initial clause weight, counting a total of 9 KENN's configurations. Clause weights are set as non-learnable parameters to force KENN to treat them all equally during the enhancement process and to ensure the presence of KENN for all the training.

5.4.2 Terminology

The following terminology will be used during the analysis:

- **F:** stands for *Father* (i.e, supertype); used to indicate a type that corresponds to a father node in the hierarchical tree
- **S:** stands for *Son* (i.e., subtype); used to indicate a type that corresponds to a child node in the hierarchical tree
- **pre-KENN state:** values of the preactivations provided to KENN as input
- **post-KENN state:** values of the preactivations after KENN (i.e., final output)

Additional note: *in some circumstances we may say that the baseline model and the pre-KENN network are equivalent. To avoid misunderstanding, the term "equivalent" is intended as identical architecture and initialization, excluding KENN's layer.*

5.4.3 Quantitative analysis

This is a high-level analysis based on the performance obtained by the models on the dev set during the training process. The involved studies will try to answer the following questions:

1. **Effects of different clause weights:** how much does the weight of a clause affect the final predictions?
2. **Effects of each KB mode:** is there a winning strategy to define clauses that brings more benefits than the others? does the winning strategy perform better than the baseline?
3. **Metrics per type:** is there a KB mode that enhances the performance on certain types?

5.4.4 Preactivations analysis

The preactivations produced by the models are the core of this analysis. Since the pre-KENN network and the baseline model share the same architecture and initialization, these studies aim to analyze how the networks evolve differently seeing the same data. Furthermore, for each KB mode, we will study how the types F and S involved in the same clause influence each other. The study can be divided into the following analysis:

1. **Distributions analysis:** it compares the distributions of the pre-KENN, post-KENN, and baseline preactivations.
2. **Finite State Machines (FSM) analysis:** it studies the effects of each KB mode with respect to the preactivations of the types F and S involved in the same clause. FSMs are a compact and intuitive way to represent the available transitions between pre-KENN and post-KENN states accordingly to the KB mode.
3. **Sankey diagrams**[15] **analysis:** it supports the FSM analysis by adding information about the number of examples involved in each transition.

The FSM analysis needs the introduction of a clear notation. In Figure 17 we can find an example of FSM with random transitions. The terminology used is quite simple:

- **FxSy**: arbitrary state where $F = x$ and $S = y$, with $x, y \in \{0, 1\}$; F and S are types involved in the same clause
- **F*S*→F*S*:** transition from a pre-KENN state to a post-KENN state
- **forbidden state:** F0S1 (circled in red); this state should never occur as final state since it represents a violation of the hierarchy

[15] a Sankey diagram is a flow diagram where the width of the arrows represents the flow rate between two states

- **p(F*S*→F*S*):** probability to move from a pre-KENN state to a post-KENN state
- **% correct:** percentage of transitions from a wrong[16] pre-KENN state to a correct[17] post-KENN state
- **% wrong:** percentage of transitions from a correct pre-KENN state to a wrong post-KENN state
- **% other:** percentage of transitions from a wrong pre-KENN state to a wrong post-KENN state

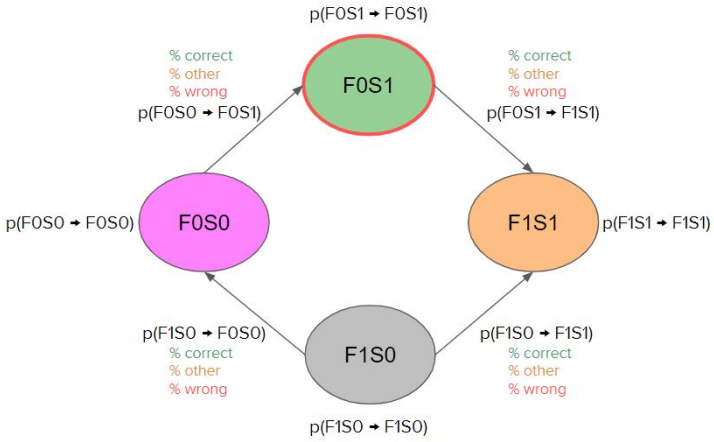


Figure 17: Example of a Finite State Machine

A few final notes on the FSM representation:

- probabilities and percentages are extracted from the dev set
- self loop transitions are omitted from the graphical representation
- the sum of probabilities of outgoing transitions, including self loops, must be equal to 1

5.4.5 Results on FIGER

Only for these experiments, a reduced version of FIGER is used with the purpose to anticipate training convergence. The subset of the dataset is extracted by a random sample that counts approximately 267K training examples.

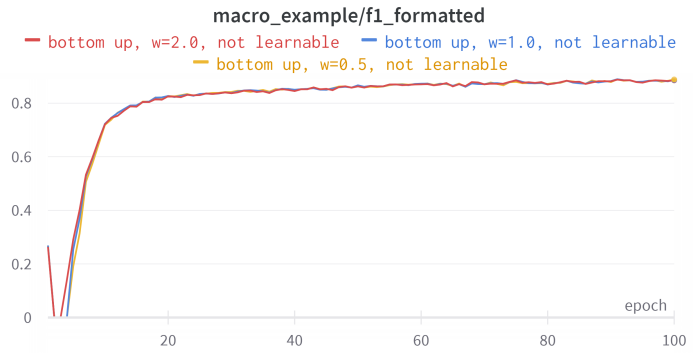
[16] wrong = at least one between F and S is wrongly predicted
[17] correct = both F and S are correctly predicted

Quantitative analysis 1 - Clause weights
The performance in terms of *macro f1 examples* obtained by varying the initial clause weights is available in Figure 18. Looking at the graphs, we can observe a common behavior: regardless of the KB mode, the influence of different weights is visible only in the first 15-20 epochs (i.e., the red line in the graph is above the others in the initial epochs). Especially for the Hybrid mode, we can see in Figure 18c how the largest weight gives a clear initial boost to the performance. However, the boost tends to diminish over the epochs until it disappears. Another interesting behavior can be found in Figure 19, which shows the evolution of the *average predictions number* in the first 20 epochs. From the figures, we can observe that the larger the weight, the higher the number of predictions, and the larger the weight, the higher the performance. We can deduce from these results that the increased performance is due to an increased number of correct predictions, otherwise we would not see an increment in the performance. To conclude, we can say that a major influence of KENN helps the network to learn faster when it has not yet seen a big amount of training examples.

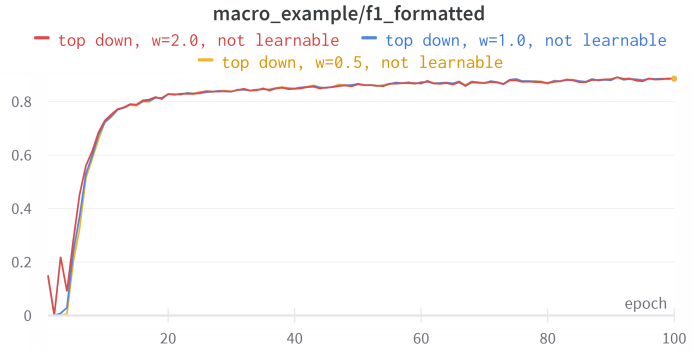
Note: *the previous considerations still apply to the other metrics not reported here*

Quantitative analysis 2 - KB modes
Moving on to the comparison between the KB modes, 3 model instances per configuration were trained by varying the random seed. The initial clause weight is set to 2.0 because in the previous experiment was the one that obtained the biggest boost. Figure 21 reports the comparison graphs in terms of *average predictions number*, *macro f1 examples* and *macro f1 classes* . Similarly to the comparison of the weight, we can draw some considerations about the first steps of the training. Starting from the *average predictions number*, we can clearly observe that the baseline model (i.e, *kb_mode:* -) produces significantly fewer predictions than KENN in the early stage. In particular, we can notice that the baseline is not able to produce any prediction for the first 4 epochs. Since the pre-KENN network and the baseline network have the same initialization, the fact that KENN requires fewer epochs to make correct predictions is due to the use of logical knowledge.

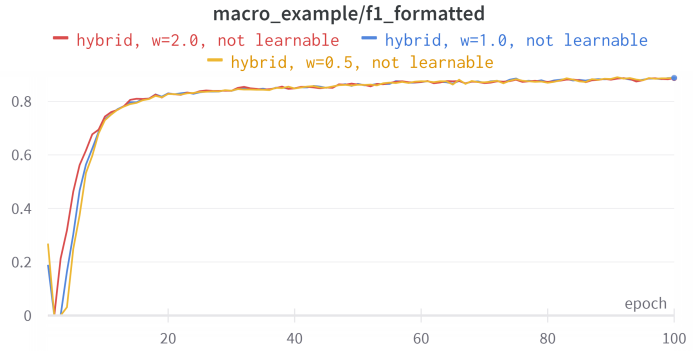
The trend observed in the *average predictions number* is also detectable by all the other metrics. Figure 20 highlights the initial boost of KENN-based models by measuring the difference with respect to the baseline in terms of *macro f1 examples*. The behavior that consolidates as the seeds vary is that Hybrid starts better than Bottom Up, which starts better than Top Down. The baseline model, instead, is the one with the slowest start. The cause of this initial ranking may be that the Hybrid mode can exploit the logical rules of both Bottom Up and Top Down modes. In the second part of the training, instead, the models converge to each other and keep improving together. The results of the final models are reported in Table 12 and are very similar for each KB mode.



(a) Bottom Up

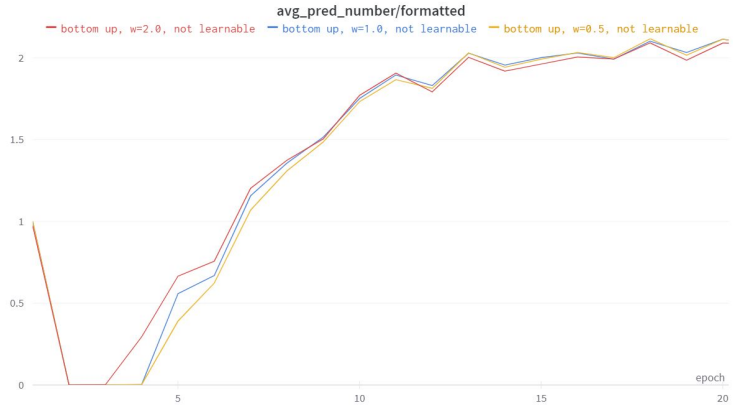


(b) Top Down

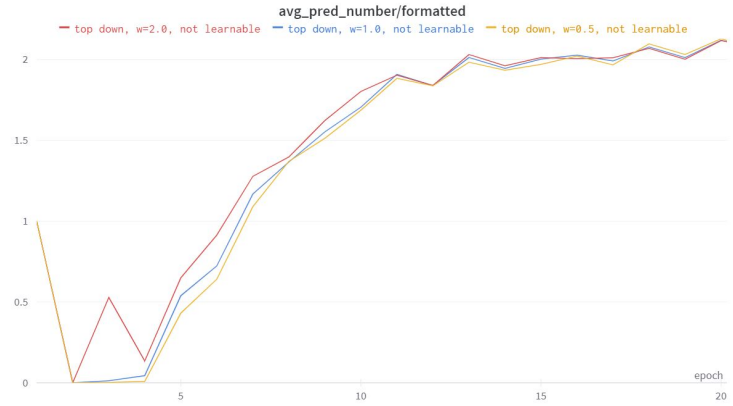


(c) Hybrid

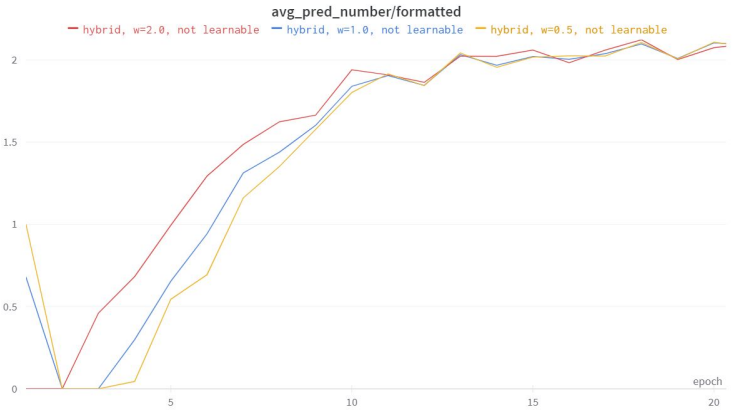
Figure 18: Comparison of *macro f1 examples* by varying the clause weight for each KB mode, with DistilBERT-based models evaluated on the dev set of FIGER.



(a) Bottom Up



(b) Top Down



(c) Hybrid

Figure 19: Comparison of *average predictions number* by varying the clause weight for each KB mode, with DistilBERT-based models evaluated on the dev set of FIGER. Zoom on the first 20 epochs.

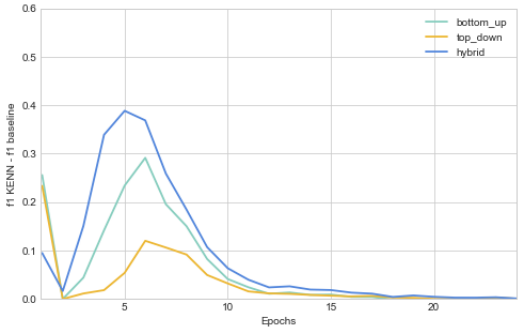


Figure 20: Difference between the *macro f1 examples* scores of KENN and baseline models with DistilBERT encoder, evaluated on the dev set of FIGER. The curves are averaged over 3 random seeds. Zoom on the first 25 epochs.

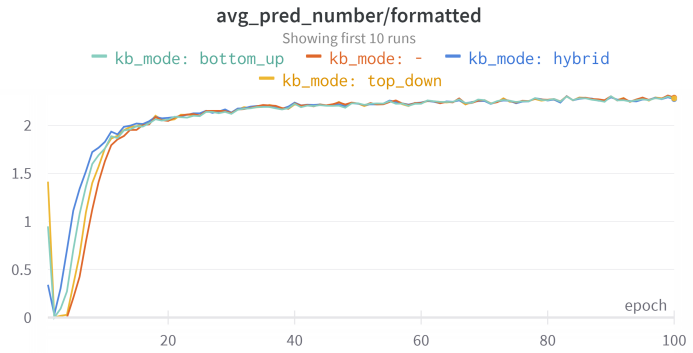
Table 12: Comparison between KENN with clause weights fixed to 2.0 and baseline model evaluated on the dev set of FIGER. Metrics at epoch 100 averaged over 3 seeds.

	Macro examples			**Macro classes**		
KB mode	**P**	**R**	**F1**	**P**	**R**	**F1**
- (baseline)	0.8944 ± 0.0009	0.8779 ± 0.0033	0.8861 ± 0.0021	0.6499 ± 0.0152	0.5943 ± 0.0051	0.6208 ± 0.0091
Bottom Up	0.8963 ± 0.0.0020	**0.8783 ± 0.0047**	**0.8872 ± 0.0020**	0.6516 ± 0.0165	**0.5986 ± 0.0110**	0.6239 ± 0.0115
Top Down	0.8944 ± 0.0008	0.8777 ± 0.0029	0.8860 ± 0.0013	0.6484 ± 0.0088	0.5913 ± 0.0112	0.6185 ± 0.0010
Hybrid	**0.8965 ± 0.0028**	0.8759 ± 0.0025	0.8861 ± 0.0017	**0.6533 ± 0.0185**	0.5974 ± 0.0050	**0.6241 ± 0.0112**

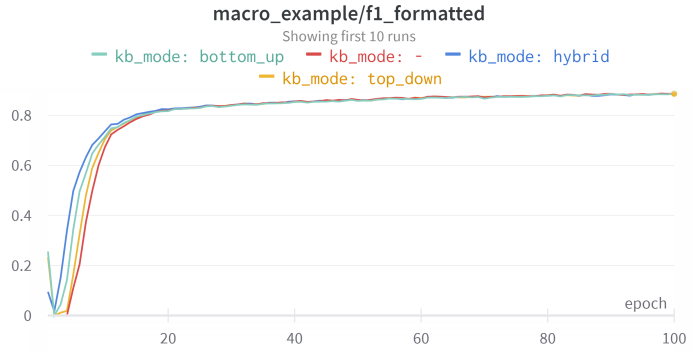
Quantitative analysis 3 - Metrics per type

We can observe some interesting behaviors by computing the metrics grouped per types F and S. Focusing on epoch 4 (i.e., the last epoch before the baseline model starts making predictions), it emerges that KENN predicts only types F. This result makes sense since an F type should be easier to predict than an S type, which is more specific. More in detail, Bottom Up and Top Down modes can lead the model to make predictions exclusively on the type `/location`, reaching an f1 score of 0.595 and 0.441 respectively. The Hybrid model, which achieves an f1 score of 0.747 on `/location`, can also predict instances of type `/person` and `/organization`.

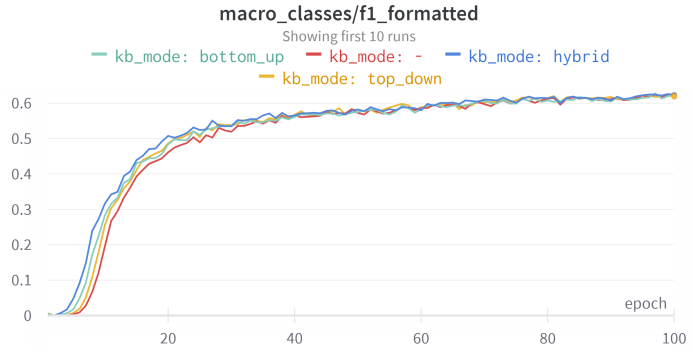
If we now focus on epoch 100, we can observe that Bottom Up performs better than Top Down on types F, with an f1 score of 0.661 and 0.646 respectively. Conversely, if we group the metrics per type S, we can observe that Top Down is better than Bottom Up, with an f1 score of 0.563 and 0.546 respectively. This behavior is consistent with the nature of the different clauses: Bottom Up propagates the information to F, Top Down to S.



(a) Average predictions number



(b) Macro f1 examples



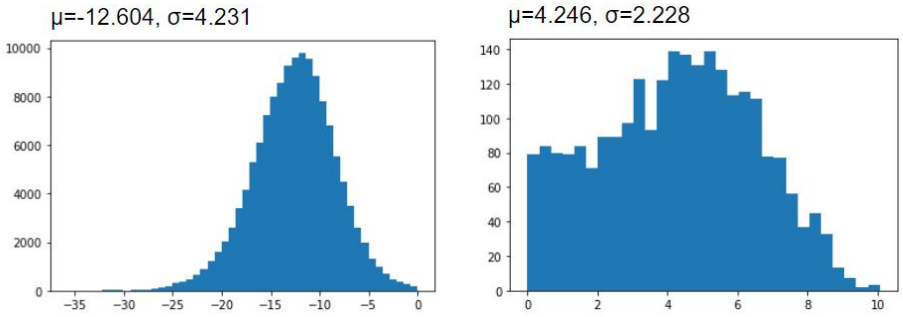
(c) Macro f1 classes

Figure 21: Comparison between KENN with clause weights fixed to 2.0 and baseline model with DistilBERT encoder, evaluated on the dev set of FIGER. Zoom on the first 20 epochs.

Quantitative analysis - Conclusion
This analysis points out that the more relevant differences are observable only at the beginning of the training. The initial gap between different clause weights is bridged after 15-20 epochs, as does the difference between the KB modes. The same behavior can be observed when comparing KENN to the baseline model. The suspicion emerging from these considerations is that, at some point of the training, the baseline model becomes as powerful as the KENN-based models. From this premises we can sketch the following hypothesis: the pre-KENN network may adapt its output to obtain some desired predictions after the knowledge enhancement. This hypothesis will be further investigated with the analysis of the preactivations.

Preactivations analysis 1 - Distributions
We concluded the quantitative analysis leaving an open question: does the pre-KENN network adapt its predictions to KENN's action? To answer this question, we can start by examining the distribution of the preactivations of every final model. Each distribution will be split into two graphs to simplify the comparison by separating positive and negative predictions using different scales. Figure 22 shows the distributions of the baseline model that will be used as a reference point. The values of x are the preactivations of each type of each example, while the y represents the frequency of a bin of preactivations.

We will start by comparing the positive predictions. In Figure 23 we can find the positive distributions of the KENN-based models. The first thing we can observe is that each KB mode produces very different pre-KENN distributions. If we look at the means (Bottom Up: 2.706, Top Down: 3.633, Hybrid: 2.227) and standard deviations (Bottom Up: 1.737, Top Down: 2.190, Hybrid: 1.434), we can also assert that they differ a lot from the baseline positive distribution (mean: 4.246, standard deviation: 2.228). On the contrary, the situation is completely different when looking at the post-KENN preactivations, since the intervention of KENN makes the distributions more similar to each other (means: 4.100 ± 0.043, standard deviations: 2.058 ± 0.038) and to the baseline. The same phenomenon can be observed by comparing the negative distribution of the baseline in Figure 22a to the pre-KENN and post-KENN negative distributions in Figure 24.



(a) Negative preactivations (b) Positive preactivations

Figure 22: Distribution of the baseline preactivations computed on the dev set of FIGER

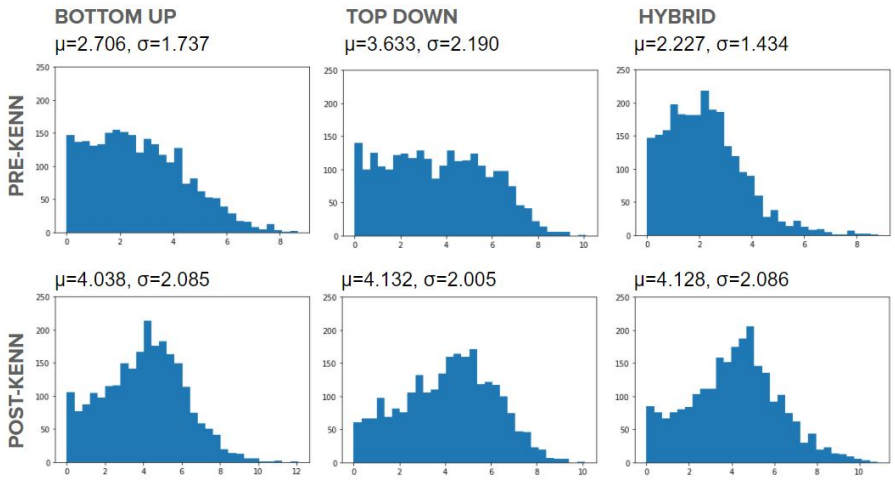


Figure 23: Distribution of the pre-KENN and post-KENN positive preactivations for each KB mode. Computed on the dev set of FIGER.

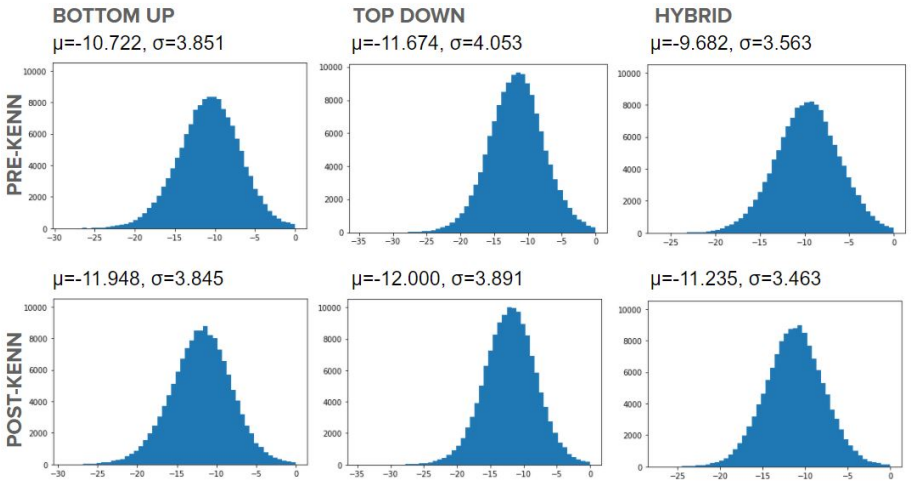


Figure 24: Distribution of the pre-KENN and post-KENN negative preactivations for each KB mode. Computed on the dev set of FIGER.

Preactivations analysis 2 and 3 - FSM and Sankey diagrams
The analysis of the Finite State Machines and the Sankey diagrams can help to understand how the pre-KENN and post-KENN preactivations are modified depending on types F and S involved in the same clause, thus providing other insights into the suspicion of adaptation emerged in the previous study.

Starting from the Bottom Up mode, we can find the FSM and the Sankey diagram in Figure 25. We remind that a Bottom Up clause will always produce a positive delta on F and a negative delta on S. Indeed, F1S0 is a sink vertex of the Bottom Up FSM because F can never be decreased and S can never be increased in a Bottom Up setup. By analyzing the information of the transitions, we can observe that:

1. The percentages of wrong transitions represent a minority.

2. The probability of remaining in the forbidden state (i.e., F0S1) is 0.03. This is reasonable because the forbidden state represents also a clause violation (i.e., 1→0) for this KB mode, so KENN will always try to modify the predictions to reach another post-KENN state. In these cases, the percentages of correctness show that the outgoing transitions from F0S1 never lead to worse predictions, but this is trivial since it is a wrong state by definition.

3. The statistics about the transition F0S1→F1S0 are quite counterintuitive since F and S are reversed correctly in 43.1% of cases. The fact that the pre-KENN network predicts F as negative and S as positive when the target is the opposite is curious. It is plausible that the pre-KENN network learned that a type F usually receives one or more positive boosts,

so it produces an initial negative prediction being aware that KENN will correct the final value.

4. The transition F0S0→F1S0 could seem weird too. It has a probability of 0.01 and a percentage of correctness of 84.3%. Some of these transitions are due to the fact that in the Bottom Up mode a type F is involved in multiple clauses, so the positive boost could derive from one or more siblings of S. However, after an investigation, it emerged that there are several examples of transitions $F0S_{1,\dots,n}0 \rightarrow F1S_{1,\dots,n}0$ where F becomes positive thanks to the aggregated boost of its negative children. In particular, on a total of 500 F0S0→F1S0 transitions, in 78 of them the effect of a single S was sufficient to enhance F. Even this behavior could be explained by the fact that the pre-KENN network learned that F is used to receive positive boosts.

The FSM and the Sankey diagram of the Top Down mode are shown in Figure 26. The first thing we can notice is that the transitions of the FSM have opposite directions with respect to Bottom Up, so the sink vertex becomes F0S1. Looking at the statistics of the transitions, we can observe that:

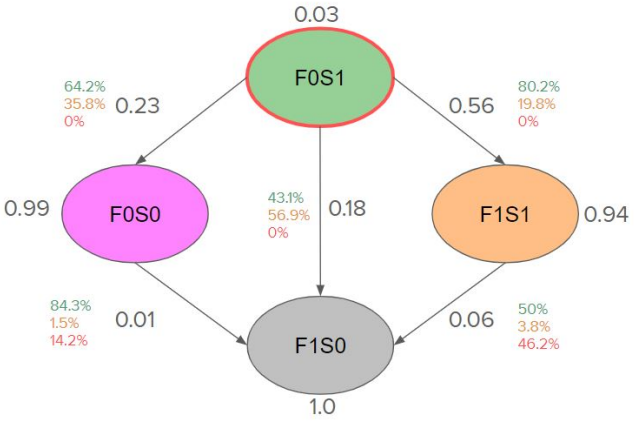
1. The percentages of wrong transitions represent a minority.

2. The incoming transitions of the forbidden state have null probabilities when starting from F0S0 or F1S1, and near-zero probability when the pre-KENN state is F1S0. This is a very interesting fact, since it means that the pre-KENN network produces predictions such that KENN cannot generate a boost that leads to the forbidden state.

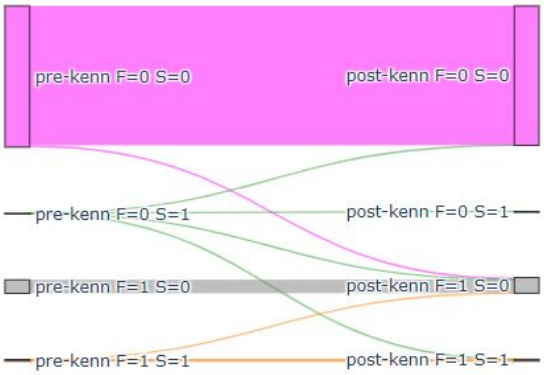
3. Even if in this KB mode the pre-KENN state F1S0 may represent a clause violation (i.e., $1 \rightarrow 0 \vee \dots \vee 0$) that KENN will try to correct, we can still find some examples of self-loop transition. Note that the high probability of 0.867 is due to the fact that in a Top Down clause the consequent is a disjunction of siblings. However, if we exclude from the count the cases in which a sibling of S is positive, we still have a high probability of 0.512 of self-loop. This means that the pre-KENN network learned to generate preactivations that prevent the effect of KENN when the desired output is F1S0.

4. If we look at the Sankey diagram in Figure 26b, we can see that the pre-KENN state F0S1 (i.e., the sink node) is missing. In this case too, the reason can be found in the fact that the pre-KENN network is aware that KENN will not be able to change the final predictions.

Finally, we can find the transitions of the Hybrid mode in Figure 27. The FSM transitions are bidirectional because they are inherited from Top Down and Bottom Up. The consequence is that there are no sink nodes and all the transitions become potentially available. The considerations we can make by observing the figures are that:

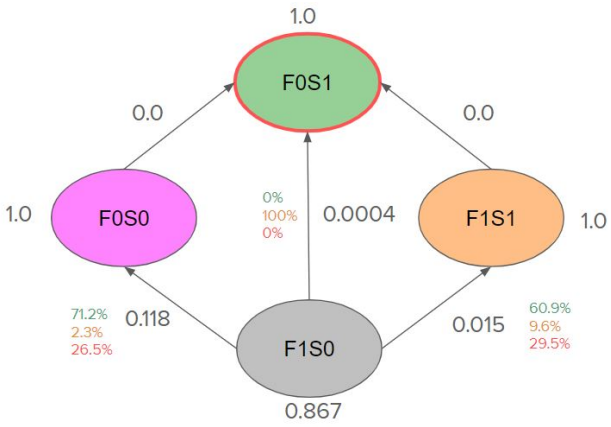


(a) FSM

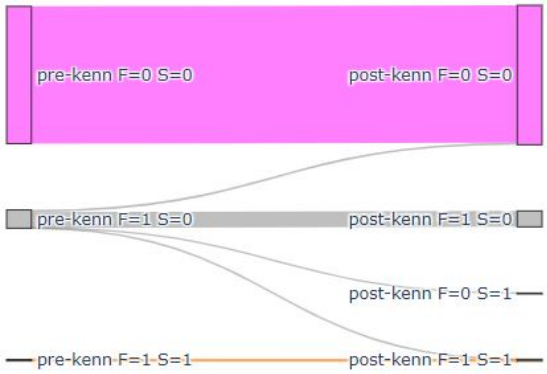


(b) Sankey diagram

Figure 25: FSM and Sankey diagram of the Bottom Up mode on the dev set of FIGER.



(a) FSM



(b) Sankey diagram

Figure 26: FSM and Sankey diagram of the Top Down mode on the dev set of FIGER.

1. The percentages of wrong transitions represent a minority.
2. Similarly to what happens in the Top Down, the probability to make a transition into the forbidden state is null.
3. The probability of remaining in the forbidden state is very low like in the Bottom Up.
4. The Sankey diagram is denser than those of Bottom Up and Top Down, since it is constituted by the mix of their transitions.

A final consideration can be done by comparing the three Sankey diagrams: even if their starting states are quite different, their final states become very similar. This fact is analogous to what we observed when comparing the pre-KENN and post-KENN preactivations.

Preactivation analysis - Conclusion
This study brought to light several aspects that confirmed the suspicion of adaptation. In the first analysis we saw that while the pre-KENN distributions had relevant differences, the post-KENN distributions became more similar to each other and to the baseline. We then detected other evident signals of adaptation by analyzing the transitions of the FSMs and the Sankey diagrams. Considering the Bottom Up mode, the most representative examples can be found in the transitions F0S1→F1S0 and F0S0→F1S0, where the pre-KENN network seems to be aware of the boost that will be produced on F by KENN. Even more interesting is the situation of the Top Down. Here, the strongest adaptation signals are given by the absence of F0S1 (i.e., sink node) as pre-KENN state in the Sankey diagram and by the high probability of not correcting a violated clause (i.e., self-loop on F1S0).

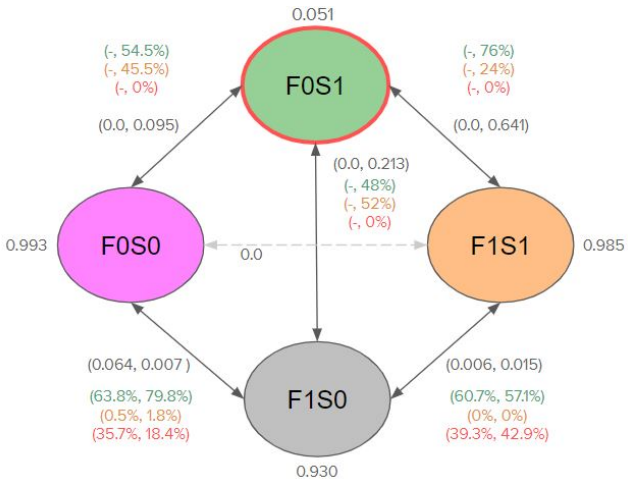
5.5 KENN with different encoders: DistilBERT vs BERT

Choosing the right language model is always important to determine the performance of the final model, especially if the goal is to compete with state-of-the-art solutions. This experiment aims to compare the behavior of KENN when dealing with networks based on encoders with different capabilities, which in our case are DistilBERT and BERT.

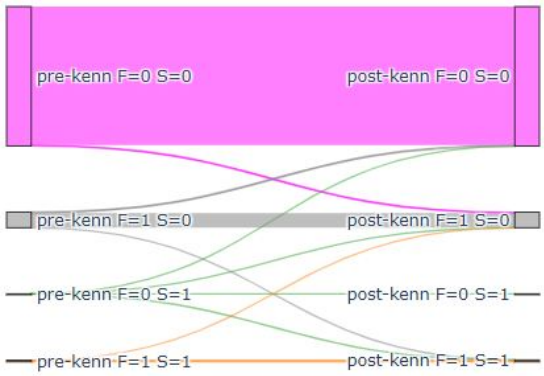
5.5.1 Setup

The models used for this experiment are trained using the *Setup B*. The experimental parameters chosen are the following:

- **KB modes:** Bottom Up, Top Down and Hybrid
- **initial clause weight:** 2.0
- **fixed clause weights**



(a) FSM - perecentages and probabilities are reported as *(x, y)*, where *x = value of down-to-up transition* and *y = value of up-to-down transition*



(b) Sankey diagram

Figure 27: FSM and Sankey diagram of the Hybrid mode on the dev set of FIGER.

- **encoder:** DistilBERT and BERT, with adapters
- **loss function:** Binary Cross-Entropy, with the weights of positive examples set to 1

This results in a total of 6 configurations. The choice of using fixed clause weights set to 2.0 comes from the previous experiments since it has been observed that higher weights led to higher boosts in the early stage of the training. The experiments are evaluated in terms of *macro f1 examples* by averaging the performance obtained from 3 random seeds per model.

5.5.2 Results on FIGER

We can start by comparing the performance obtained by the baseline model with different encoders. Looking at the graph in Figure 28, we can observe a significant gap between the models. The difference, which is larger at the beginning, persists for the duration of the training and shows the major capabilities of BERT. The final models reach a score of 0.9155 and 0.8911 for BERT and DistilBERT, respectively.

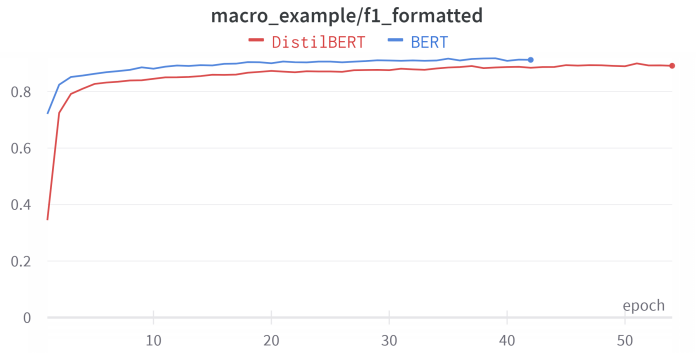


Figure 28: Baseline DistilBERT vs Baseline BERT in terms of *macro f1 examples* on the dev set of FIGER

Now that the superiority of BERT has been confirmed, we can proceed with the comparison between the baseline and KENN-based models. In Figures 29 and 30 are presented the performance obtained with DistilBERT and BERT, respectively. As we can see, the behavior of KENN is completely different depending on the encoder. Starting from DistilBERT, the initial boost given by KENN is clearly visible, especially when comparing the baseline to the Hybrid model. However, as already discussed in the quantitative analysis, this boost vanishes in the rest of the training when the models converge. When using BERT, instead, the baseline model is the one that has the best start. While Top Down has a similar training trend even at the beginning, Bottom Up and Hybrid are always inferior.

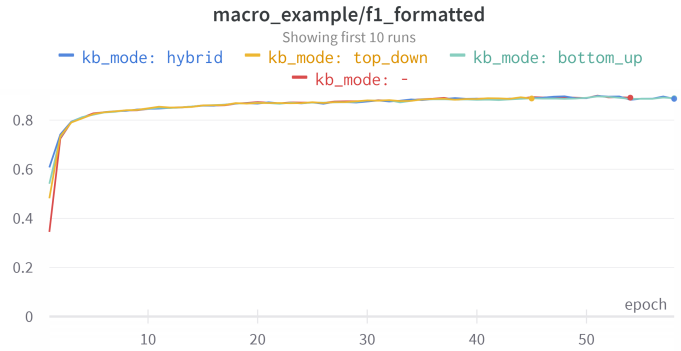


Figure 29: Baseline DistilBERT vs KENN DistilBERT in terms of *macro f1 examples* on the dev set of FIGER.

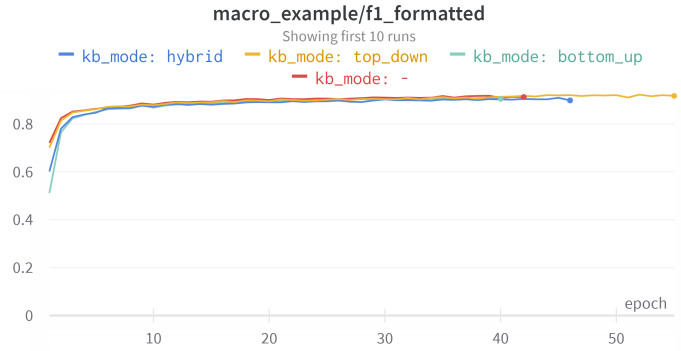


Figure 30: Baseline BERT vs KENN BERT in terms of *macro f1 examples* on the dev set of FIGER.

5.5.3 Results on BBN

Figure 31 compares DistilBERT and BERT when using the baseline model. Even this dataset confirmed the superiority of BERT, which reached the score of 0.9047, while DistilBERT's score is 0.8845.

The trends of KENN-based models compared to the baseline when using DistilBERT and BERT are available in Figures 32 and 33, respectively. Looking at the graphs, we can make the same observation done for FIGER: the logical knowledge gives an initial boost only when using DistilBERT. Moving on to BERT, except Bottom Up, which has a slower start, the other models have very similar training trends.

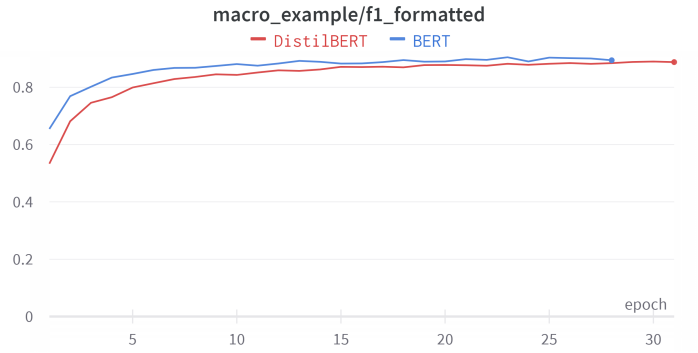


Figure 31: Baseline DistilBERT vs Baseline BERT in terms of *macro f1 examples* on the dev set of BBN.

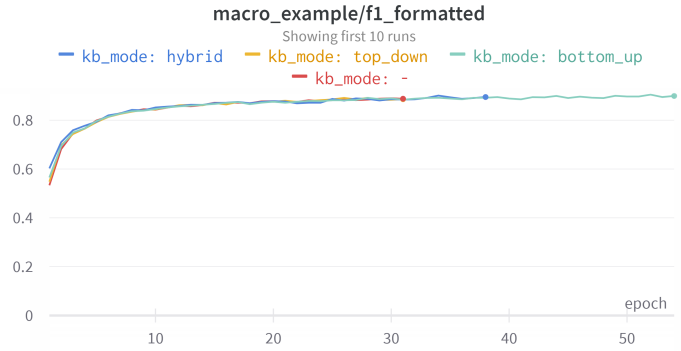


Figure 32: Baseline DistilBERT vs KENN DistilBERT in terms of *macro f1 examples* on the dev set of BBN.

5.5.4 Conclusion

The results obtained on the two datasets highlighted that the benefits of the logical knowledge are visible only in the early stage of the DistilBERT-based models. The fact that KENN worsened the performance of the BERT-based models was quite unexpected. Except for the Top Down mode, it seems that the injection of logical knowledge disturbed the learning process of the other models, especially at the beginning of the training. An explanation for this behavior can be found in the words of KENN's authors, which said that KENN has more difficulties when integrated into neural networks that are already capable to satisfy the provided clauses, since any bias is introduced towards their satisfaction [9].

In our case, we observed that the performance of the baseline model was

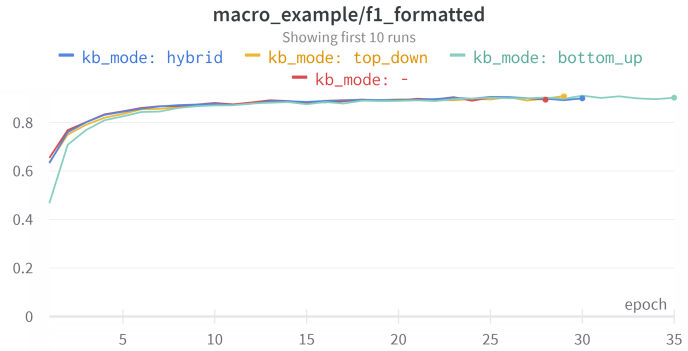


Figure 33: Baseline BERT vs KENN BERT in terms of *macro f1 examples* on the dev set of BBN.

substantially improved when using BERT. For this reason, it may be possible that BERT, which provides a richer text representation than DistilBERT, is already able to implicitly learn from data the hierarchical information without the need for logical knowledge. Furthermore, we can think about the differences between the KB modes to explain why Top Down has higher performance than Bottom Up: while the knowledge provided by a Bottom Up clause could be learned from data (i.e., the model learns that a subtype always co-occur with its supertype), Top Down clauses could introduce more bias into the learning process, thus adding new solutions to the Hypothesis Space as said in [9].

5.6 KENN with multiloss function

In previous experiments, it has been observed that the pre-KENN network adapts to the presence of KENN regardless of the parameters configuration. From these results comes the intuition to make the pre-KENN network more independent, with the purpose of exploiting the logical knowledge more effectively. To achieve this, the idea is to define a custom multiloss function to simultaneously improve the quality of the pre-KENN and post-KENN predictions.

The proposed multiloss function is computed by combining the Binary Cross-Entropy (BCE) loss values obtained from the pre-KENN and post-KENN pre-activations. To make it possible, the model has been adapted to provide two outputs: one before and one after KENN's layer. The goal is to optimize the post-KENN predictions while preserving a discrete quality of pre-KENN predictions. The final loss is computed by the convex combination of the two losses whose influences are regulated by the parameter α. The resulting formula is the following:

$$L(Y, Y', Y_t) = \alpha \cdot BCE(Y, Y_t) + (1 - \alpha) \cdot BCE(Y', Y_t)$$

where Y denotes the pre-KENN predictions, Y' the post-KENN predictions, and Y_t the values of the ground truth.

5.6.1 Setup

The models involved in this experiment are trained following the *Setup B*. The configuration of the other parameters is the following:

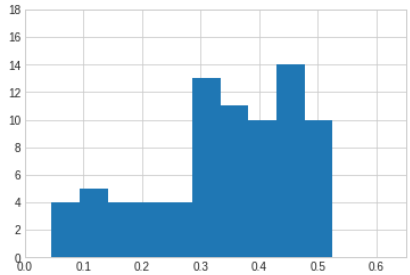
- **KB modes:** Bottom Up and Top Down
- **initial clause weights:** 0.5 and variable[18]
- **learnable clause weights**
- **encoder:** DistilBERT and BERT, with adapters
- **loss function:** multiloss with $\alpha = 0.5$

The total number of configurations obtained by varying these parameters is 8. Since the pre-KENN network is expected to be less influenced by KENN, the clause weights are set as learnable parameters with small initial values to start with a soft influence and let the network establish the relevance of each clause. Indeed, the choice of learnable clause weights allows us to study the weight evolution during the epochs and determine which are the most useful and the least useful clauses. Furthermore, for each clause, the types that are the antecedents of an implication rule (i.e., the types that propagate the information and uniquely characterize the used KBs) have been studied to look for some recurring behavioral patterns. The *Hybrid* mode is excluded from this study because it involves the same clauses of *Bottom Up* and *Top Down*, but with the addition of noise due to the presence of conflicts. For this reason, it would not have been possible to carry out an accurate analysis.

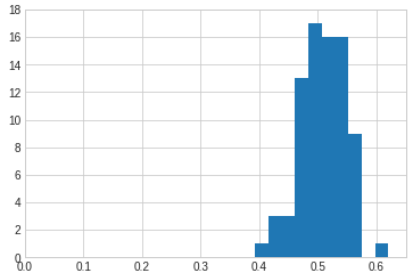
5.6.2 Results on FIGER

DistilBERT The results obtained using DistilBERT as encoder show some very interesting behaviors of the clause weights. Starting from the Bottom Up, if we look at the distribution of final clause weights in Figure 34a we can see that almost every weight decreases its initial value. This attitude is not observable in Figure 34b when using KENN with a standard loss and learnable weights. If we look at the figure, we can see widely different results since almost every weight increases its starting value. The same trend is detectable in the Top Down mode, as shown in Figure 35.

[18] With the term "variable" we mean that each clause can be initialized with a different value. The original implementation of KENN did not contemplate this possibility when setting clause weights as learnable parameters, so we introduced a small modification to make it possible.

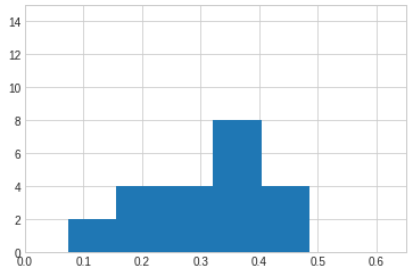


(a) Multiloss model - Epoch 55 (563K examples)

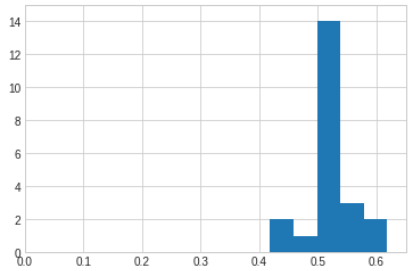


(b) Standard loss model - Epoch 42 (430K examples)

Figure 34: Distributions of the final learned clause weights for the Bottom Up KB mode using DistilBERT-based models - FIGER



(a) Multiloss model - Epoch 42 (430K examples)

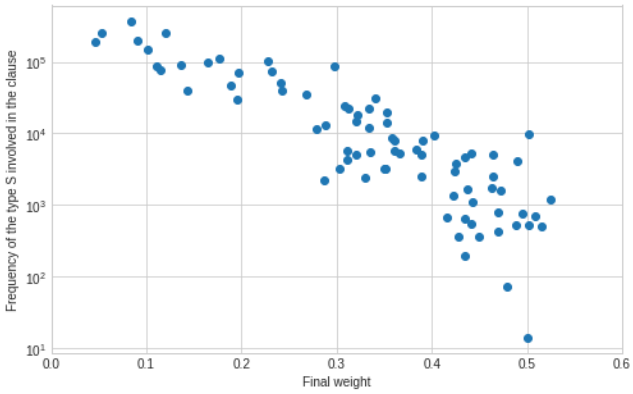


(b) Standard loss model - Epoch 39 (399K examples)

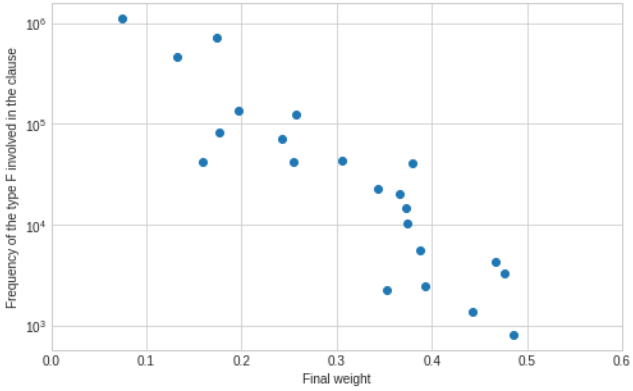
Figure 35: Distributions of the final learned clause weights for the Top Down KB mode using DistilBERT-based models - FIGER

Moving on to the analysis of the types involved in the clauses we can observe an unexpected behavior. For both Bottom Up and Top Down modes, this study intercepts a remarkable fact: the more a type occurs in the training set, the lower the final weight of its clause. The few clauses that preserve a high weight are those whose antecedents are the rarest in the dataset. Starting from the Bottom Up mode, we can see in Figure 36a a graph showing the relation between the final weight and the frequency of each type S. Looking at the figure, we can clearly observe that there is a strong correlation between weight and frequency. Indeed, the correlation coefficient is -0.77. The Top Down mode (Figure 36b) also presents a negative correlation, this time of -0.68 with respect to the frequencies of types F.

By inspecting the weight evolution over the epochs, it emerged that the clause weights keep decreasing until the last epoch, but it is not known for how long they would have continued. To observe in advance the effect of higher epochs and see if some clauses would have reached a weight close to zero, new



(a) Bottom Up - Epoch 55 (563K examples)



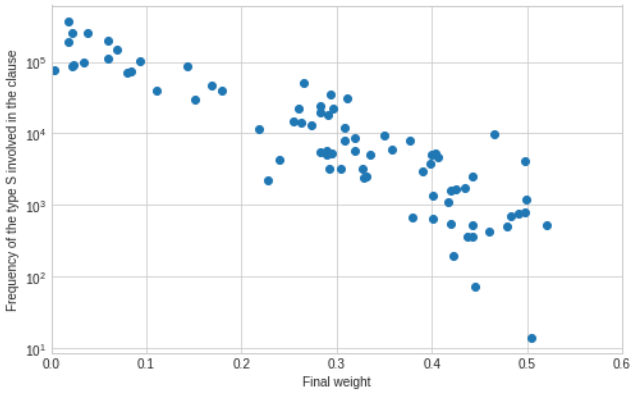
(b) Top Down - Epoch 42 (430K examples)

Figure 36: Relation between types frequency (log scale) and final clause weights of multiloss models using DistilBERT - FIGER

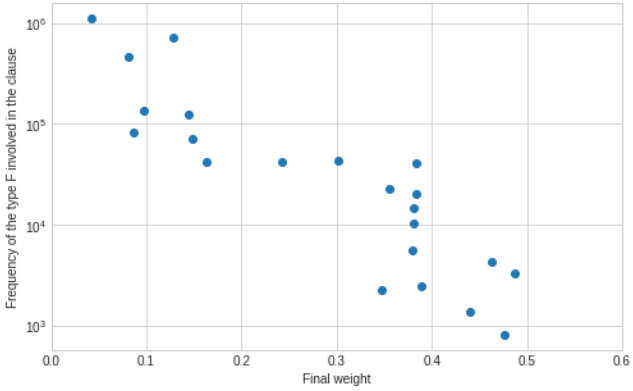
models were trained starting from variable weights assigned with respect to the frequency of the antecedents of each clause. Clauses involving popular types were assigned a weight of 0.2, while the others kept a weight of 0.5. The weight assignment is based on a frequency threshold of 70K for Bottom Up and 50K for Top Down[19]. The results of these runs are shown in Figure 37a and Figure 37b for Bottom Up and Top Down, respectively. As we can see, most clause weights decreased their values even when starting from lower initial values.

BERT The experiments performed using BERT showed an identical behavior. The results obtained with weights set to 0.5 for every clause are shown

[19]the values are chosen by observing the graphs, without formalizing a mathematical criterion



(a) Bottom Up - Epoch 71 (727K examples)



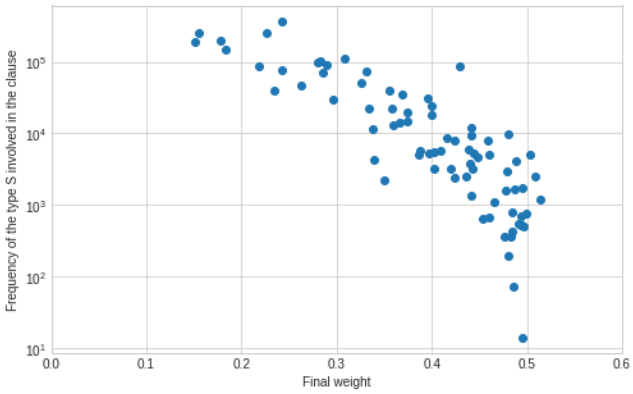
(b) Top Down - Epoch 42 (430K examples)

Figure 37: Relation between types frequency (log scale) and final clause weights of multiloss models using DistilBERT and *variable* clause weights - FIGER

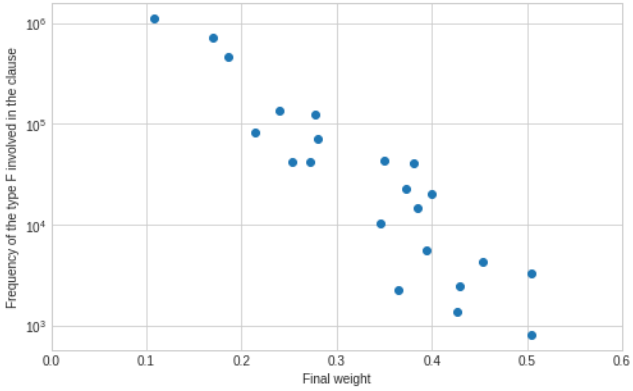
in Figure 38. The correlation between weight and frequency is similar (-0.76 in Bottom Up, -0.74 in Top Down) as well as the weight evolution during the epochs. The only difference with respect to DistilBERT is that BERT needs fewer epochs to converge, so its weights have decreased less as it performed fewer backward propagation operations. Much lower weights are reached when using the setup with variable weights as reported in Figure 39.

5.6.3 Results on BBN

The experiments on BBN were performed directly on BERT. In Figure 40 are represented the graphs of weights and frequencies using the same weight for each clause. The correlation coefficients of the Bottom Up and Top Down modes are -0.73 and -0.55, respectively. The results obtained after repeating



(a) Bottom Up - Epoch 35 (358K training examples)



(b) Top Down - Epoch 35 (358K training examples)

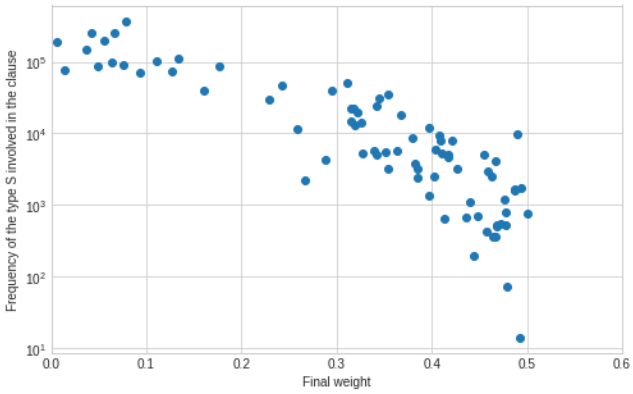
Figure 38: Relation between types frequency (log scale) and final clause weights of multiloss models using BERT - FIGER

the experiments using variable weights are shown in Figure 41. The weight assignment is based on a frequency threshold of 1700 for Bottom Up and 1500 for Top Down[20]. Looking at the graphs, it is possible to observe that most of the weights continued to decrease.

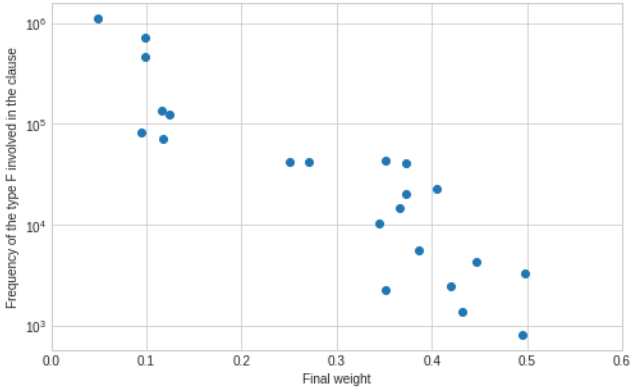
5.6.4 Conclusion

The multiloss experiments showed that the pre-KENN network suffers the presence of some logical clauses when it is forced to not adapt its predictions to KENN. In particular, it has been possible to see that the learning process pe-

[20]the values are chosen by observing the graphs, without formalizing a mathematical criterion



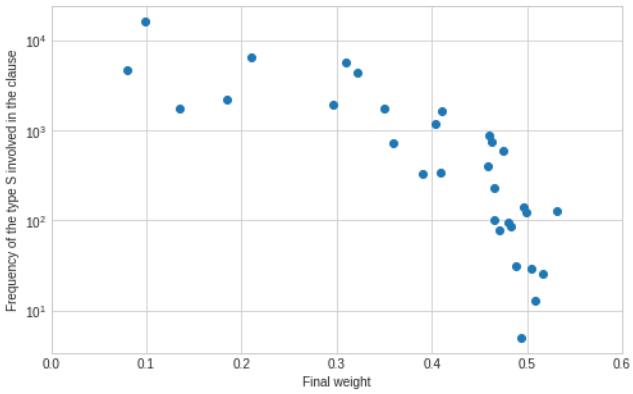
(a) Bottom Up - Epoch 49 (501K training examples)



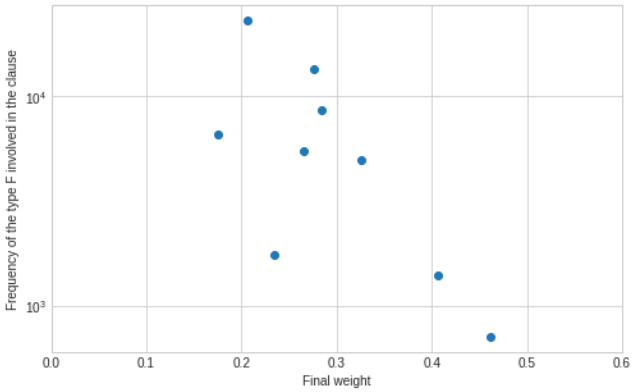
(b) Top Down - Epoch 33 (337K training examples)

Figure 39: Relation between types frequency (log scale) and final clause weights of multiloss models using BERT and *variable* clause weights - FIGER

nalized clauses involving the most frequent types much more than the others. This behavior can be motivated by the fact that if a network sees a lot of instances labeled with some types, then it becomes able to correctly classify them without needing logical knowledge. For this reason, it seems that the action of KENN on frequent types is seen as noise added to the final predictions. Conversely, the enhancement of rarer types is not penalized by the learning process, so logical clauses seem to be helpful for the network when few examples labeled with those types are available. For this reason, using a multiloss setup could be a reasonable choice to exploit logical knowledge only where needed, thus avoiding having side effects on the types that are easier to predict.

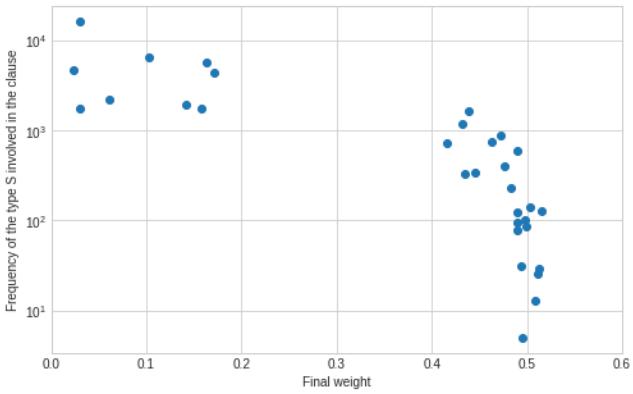


(a) Bottom Up - Epoch 28 (286K training examples)

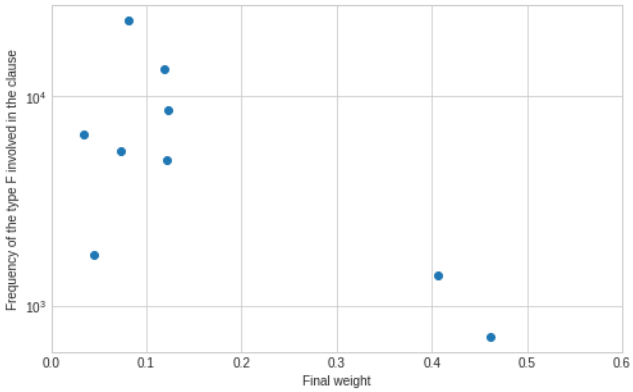


(b) Top Down - Epoch 19 (194K training examples)

Figure 40: Relation between types frequency (log scale) and final clause weights of multiloss models using BERT - BBN



(a) Bottom Up - Epoch 19 (194K training examples)



(b) Top Down - Epoch 19 (194K training examples)

Figure 41: Relation between types frequency (log scale) and final clause weights of multiloss models using BERT and *variable* clause weights - BBN

5.7 Test set evaluation

The models chosen for the test set evaluation come from the multiloss experiments. Based on *Setup B*, they have the following configuration:

- **KB modes:** Bottom Up and Top Down
- **initial clause weights:** 0.5 and variable
- **learnable clause weights**
- **encoder:** BERT with adapters
- **loss function:** multiloss with $\alpha = 0.5$

The choice of the multiloss setup has the following reasons derived from the experimental results:

1. When using a standard loss setup, the pre-KENN network adapts its predictions to the presence of KENN and achieves results that are very similar to the baseline ones. This means that KENN-based models can be seen as simple baseline models that have had different evolutions due to the noise introduced by KENN.
2. When using a multiloss setup, the pre-KENN network avoids the adaptation thanks to the pre-KENN loss. The final clause weights will ensure that the network only uses KENN when it really needs it.
3. The choice of variable clause weights may emphasize the effect of the multiloss on final weights, thus producing models in which the influence of KENN is slightly different.

The test set has been evaluated using the metrics of *macro f1 examples* and *macro f1 classes*. The inference rules used to compute the final predictions are the *threshold* and *threshold or max*.

Results on FIGER

Table 13 shows the results obtained on FIGER. If we look at the performance with the *threshold* inference, we can see that the Top Down mode is the best in every metric. With respect to the baseline model, it gains 0.0151 on *macro f1 examples* and 0.0108 on *macro f1 classes*.

KENN-based models become even more competitive when using the *threshold or max* inference. In this case, the metrics obtained by Bottom Up and Top Down are quite similar. The best performance on *macro f1 examples* and *macro f1 classes* are reached by Bottom Up and Top Down by gaining 0.0181 and 0.0137, respectively, over the baseline. Note that by changing the inference rule, Bottom Up and Top Down increase their *macro examples f1* much more than the baseline.

Table 13: Performance of the baseline and KENN multiloss models on the test set - FIGER

		Macro examples			Macro classes		
KB mode	**Inference**	**P**	**R**	**F1**	**P**	**R**	**F1**
- (baseline)	threshold	0.7884	0.8601	0.8227	0.2305	0.2526	0.2410
Bottom Up (variable weights)	threshold	0.7631	0.8501	0.8043	0.2171	0.2459	0.2306
Bottom Up	threshold	0.7854	0.8611	0.8215	0.2343	0.2630	0.2478
Top Down (variable weights)	threshold	0.7560	0.8627	0.8058	0.2157	0.2445	0.2292
Top Down	threshold	**0.7997**	**0.8796**	**0.8378**	**0.2404**	**0.2643**	**0.2518**
- (baseline)	threshold or max	0.8008	0.8699	0.8339	0.2355	0.2542	0.2445
Bottom Up (variable weights)	threshold or max	0.7950	0.8795	0.8351	0.2156	0.2492	0.2312
Bottom Up	threshold or max	**0.8174**	0.8898	**0.8520**	0.2327	**0.2718**	0.2507
Top Down (variable weights)	threshold or max	0.7773	0.8831	0.8268	0.2298	0.2518	0.2403
Top Down	threshold or max	0.8139	**0.8903**	0.8504	**0.2495**	0.2676	**0.2582**

Results on BBN

In Table 14 we can find the results obtained on BBN. Except for the *macro classes* metrics, the best results are achieved by both the Top Down modes regardless of the inference rule. This time, in constrast to FIGER, the baseline and KENN-based models have a similar boost on the performance when moving from *threshold* to *threshold or max* inference.

Table 14: Performance of the baseline and KENN multiloss models on the test set - BBN

		Macro examples			Macro classes		
KB mode	**Inference**	**P**	**R**	**F1**	**P**	**R**	**F1**
- (baseline)	threshold	0.7428	0.8197	0.7793	**0.4893**	0.4981	**0.4937**
Bottom Up (variable weights)	threshold	0.7489	0.8222	0.7839	0.4783	0.4947	0.4864
Bottom Up	threshold	0.7417	0.8149	0.7766	0.4682	**0.5099**	0.4882
Top Down (variable weights)	threshold	0.7540	0.8279	0.7892	0.4825	0.4990	0.4906
Top Down	threshold	**0.7551**	**0.8297**	**0.7906**	0.4560	0.4770	0.4663
- (baseline)	threshold or max	0.7531	0.8272	0.7884	0.4730	0.5007	0.4865
Bottom Up (variable weights)	threshold or max	0.7617	0.8305	0.7946	0.4607	0.5019	0.4804
Bottom Up	threshold or max	0.7516	0.8216	0.7851	0.4589	0.5125	0.4842
Top Down (variable weights)	threshold or max	0.7649	0.8353	0.7986	**0.4775**	**0.5131**	**0.4947**
Top Down	threshold or max	**0.7676**	**0.8387**	**0.8016**	0.4477	0.4917	0.4687

Test set considerations

The configurations involving the Top Down mode are the ones that achieved the best results. If we group the results of both datasets by the inference rule and consider the performance of Top Down with and without variable weights together, we can see that this KB mode reached the highest metric in 18 out of 24 cases[21]. The reason for these results may be that is known that the classification of a supertype (i.e., the antecedent of a Top Down clause) is generally easier than the classification of a subtype, so the boost produced by KENN seems to be more effective when propagated from supertypes to subtypes. However, this aspect needs to be further investigated and has been left for future work.

A final consideration can be made on the test sets as they have significant differences compared to the training sets (section 4.1.3). The noise that characterizes a dataset generated by distant supervision techniques could limit the potential of a model. By inspecting some automatically annotated entries from the training sets, we can find several mentions labeled with completely different types even if they appear in very similar contexts. Conversely, this kind of entries are not present in the test sets. This means that a model trained on this noisy data could have difficulties to learn some types. Regarding KENN, this noise could be even more confusing due to the presence of logical clauses, which have an important role in the learning process. If we think about the multiloss experiments, for example, a cleaner dataset could have led to a completely different evolution of clause weights, thus leading to a different influence of KENN at inference time.

[21] $|metrics| \times |inference\ rules| \times |datasets| = 6 \times 2 \times 2 = 24$ metrics

5.8 Impact of KENN on costs

This last experiment aims to verify if KENN has a strong impact on the costs of the neural network in which it is integrated. The models used to carry out this study are those used to perform the test set evaluation. For this experiment, it is not necessary to vary the dataset, the KB mode, or other KENN parameters since they do not affect the results.

First of all, we can make some considerations about the additional resources required by KENN-based models. When KENN's layer is added to a model, the network increases its number of parameters accordingly to the number of clauses contained in the logical KB. If we consider that the baseline model with DistilBERT counts about 66M network parameters, the number of clauses becomes irrelevant when compared to the total number of parameters. The same is especially true for BERT, since it has around 340 million parameters.

Moving on to the costs in terms of inference time, the models have been tested on a batch of 64 examples. To slow down the inference time and better observe differences even using small data, the models were not loaded in GPU. For each model, prediction times were measured over 10 runs. The results are shown in Table 15. Even in this case, we can see that the impact of KENN on the costs of a network is negligible since the inference time is much more influenced by the complexity of the encoders.

Table 15: Comparison between inference time of the baseline and KENN-based models on a batch of 64 examples without using GPU

	DistilBERT	**BERT**
baseline	2.217s (± 0.166s)	15.150s (± 0.297s)
KENN	2.300s (± 0.144s)	15.915s (± 0.250s)

6 Conclusion

The focus of this thesis was to understand how to exploit the hierarchical information of a Fine-grained Entity Typing dataset by using the Neuro-Symbolic Integration approach KENN.

In section 1.2, the following research question have been defined:

- ***RQ1:*** *How can we express hierarchical information through logical rules?*
- ***RQ2:*** *How does KENN behave with different configurations?*
- ***RQ3:*** *What are the benefits of using KENN in FET?*

Starting from the *RQ1*, two main groups of constraints that could be derived from a hierarchy have been identified: *vertical constraints*, to express specialization relations, and *horizontal constraints* to express the disjointness or equivalence between two types. For each group, different strategies to define a logical KB have been proposed. All these strategies were formalized through FOL formulas, then analyzed with respect to their applicability in KENN. Thanks to the FOL formulation, each strategy can be generalized and adapted to be used in other NSI approaches. Among the proposals, the strategies that have been tested and deepened with KENN are those that impose vertical constraints: Bottom Up, Top Down, and Hybrid.

To answer to the *RQ2*, many experiments have been carried out to study the behavior of KENN. We observed in the first experiments that, regardless of the configuration of KENN's parameters, the pre-KENN network tends to adapt its predictions to the presence of KENN to obtain some desired output. Even more interesting are the results obtained in the multiloss experiments, since we proved that the pre-KENN network suffers from the presence of some logical clauses when it is forced to not adapt. In particular, the results showed that the learning process strove to drastically reduce the influence of unnecessary clauses. In light of these results, the best setup seems to be the one with the multiloss function and learnable weights, since with this setup each clause influences the final predictions depending on its real usefulness. Moreover, the results suggest the most effective KB mode is Top Down.

Finally, we have the *RQ3*, which would need more experiments to find an exhaustive answer. If we limit to the experiments of this thesis, we can try to give an answer starting from these two considerations:

1. DistilBERT's experiments showed that KENN enhanced the learning process in the early epochs when models had not yet seen many examples.
2. The learned clause weights of the multiloss setup showed that the network considers helpful the clauses involving the rarest types of the dataset, while suffering the presence of the clauses involving frequent types.

These results suggest that KENN may be more beneficial in resource-limited applications, where only low-capability models or small data are available. Moreover, the authors of KENN showed in [8, 9] that their approach outperformed

many state-of-the-art solutions in zero-shot learning, thus confirming that logical knowledge could be really useful to learn from a few data. This fact is very promising for future works, since KENN could be applied in a scenario where a new fine-grained type is added to the hierarchy of the type set, but only a few examples can be provided to the model to learn to classify the new type. As shown in the last experiment on the impact of KENN on costs, this solution would not be expensive since the additional costs introduced by KENN are negligible.

These research questions could be further deepened and extended to new ones, depending on the results of the future works proposed in the next section.

7 Future works

This thesis can be extended with many other studies. Some interesting experiments and analysis that could be performed starting from this work are:

1. **In-depth analysis of multiloss models:** Investigate on the results shown in section 5.7 to understand which are the real benefits brought by KENN.

2. **Denoised datasets:** Repeat these experiments by using the same datasets after applying a strategy of denoising to see if the logical knowledge would be exploited more effectively.

3. **T-conorm:** Try different t-conorm implementations (e.g., Łukasiewicz, Product) to see how they influence the behavior of KENN.

4. **`/*/other`:** Extend the type set of the datasets by adding to each supertype of the hierarchy the subtype `/*/other`. In this way, each example is forced to have only full-path annotation, thus making the Top Down clauses always consistent with the Open World Assumption of the original datasets.

5. **Horizontal constraints:** Try to exploit the proposed horizontal constraints to build richer logical KBs.

6. **Transfer learning:** Use KENN in a *domain adaptation* scenario to map the learned types of a source domain to new types of a similar target domain. In this way, the training of the model on the target dataset can exploit the information learned from the source dataset.

7. **Test the introduction of new types after training:** Use KENN to retrain a model when a new fine-grained type is added to the type set. In this case, logical knowledge may help to learn the new type by using only a few examples.

8. **Test other NSI approaches:** Using the same logical knowledge, verify if other NSI approaches would obtain the same results of KENN.

8 References

[1] "Appendix C: Named entity task definition (v2.1)," in *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995*, 1995. [Online]. Available: https://aclanthology.org/M95-1024

[2] M. Fleischman, "Automated subcategorization of named entities," in *ACL*, 2001.

[3] M. Fleischman and E. Hovy, "Fine grained classification of named entities," in *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. [Online]. Available: https://aclanthology.org/C02-1130

[4] X. Ling and D. S. Weld, "Fine-grained entity recognition," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5152

[5] R. Weischedel and A. Brunstein, "Bbn pronoun coreference and entity type corpus," *Linguistic Data Consortium, Philadelphia*, vol. 112, 2005.

[6] D. Gillick, N. Lazic, K. Ganchev, J. Kirchner, and D. Huynh, "Context-dependent fine-grained entity type tagging," *arXiv preprint arXiv:1412.1820*, 2014. [Online]. Available: https://arxiv.org/abs/1412.1820

[7] M. K. Sarker, L. Zhou, A. Eberhart, and P. Hitzler, "Neuro-symbolic artificial intelligence," *AI Communications*, no. Preprint, pp. 1–13, 2021.

[8] A. Daniele and L. Serafini, "Knowledge enhanced neural networks," in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2019, pp. 542–554.

[9] ——, "Neural networks enhancement with logical knowledge," *arXiv preprint arXiv:2009.06087*, 2021.

[10] K. Babić, S. Martinčić-Ipšić, and A. Meštrović, "Survey of neural text representation models," *Information*, vol. 11, no. 11, p. 511, 2020.

[11] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[12] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. [Online]. Available: https://aclanthology.org/D14-1162

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[15] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[17] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[18] J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder, "MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 7654–7673. [Online]. Available: https://aclanthology.org/2020.emnlp-main.617

[19] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, "AdapterFusion: Non-destructive task composition for transfer learning," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, 2021, pp. 487–503. [Online]. Available: https://aclanthology.org/2021.eacl-main.39

[20] F. López and M. Strube, "A fully hyperbolic neural model for hierarchical multi-class classification," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association

for Computational Linguistics, 2020, pp. 460–475. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.42

[21] M. A. Yosef, S. Bauer, J. Hoffart, M. Spaniol, and G. Weikum, "HYENA: Hierarchical type classification for entity names," in *Proceedings of COLING 2012: Posters*. Mumbai, India: The COLING 2012 Organizing Committee, Dec. 2012, pp. 1361–1370. [Online]. Available: https://aclanthology.org/C12-2133

[22] S. Shimaoka, P. Stenetorp, K. Inui, and S. Riedel, "An attentive neural architecture for fine-grained entity type classification," in *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*. San Diego, CA: Association for Computational Linguistics, 2016, pp. 69–74. [Online]. Available: https://aclanthology.org/W16-1313

[23] H. Li, C.-Y. Lin, M. Osborne, G. G. Lee, and J. C. Park, Eds., *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea: Association for Computational Linguistics, 2012. [Online]. Available: https://aclanthology.org/P12-1000

[24] P. Xu and D. Barbosa, "Neural fine-grained entity type classification with hierarchy-aware loss," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 16–25. [Online]. Available: https://aclanthology.org/N18-1002

[25] D. Yogatama, D. Gillick, and N. Lazic, "Embedding methods for fine grained entity type classification," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, 2015, pp. 291–296. [Online]. Available: https://aclanthology.org/P15-2048

[26] A. Abhishek, A. Anand, and A. Awekar, "Fine-grained entity type classification by jointly learning representations and label embeddings," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 797–807. [Online]. Available: https://aclanthology.org/E17-1075

[27] F. López, B. Heinzerling, and M. Strube, "Fine-grained entity typing in hyperbolic space," in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 169–180. [Online]. Available: https://aclanthology.org/W19-4319

[28] X. Ren, W. He, M. Qu, L. Huang, H. Ji, and J. Han, "AFET: Automatic fine-grained entity typing by hierarchical partial-label embedding," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, 2016, pp. 1369–1378. [Online]. Available: https://aclanthology.org/D16-1144

[29] X. Ren, W. He, M. Qu, C. R. Voss, H. Ji, and J. Han, "Label noise reduction in entity typing by heterogeneous partial-label embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds. ACM, 2016, pp. 1825–1834. [Online]. Available: https://doi.org/10.1145/2939672.2939822

[30] A. Rahman and V. Ng, "Inducing fine-grained semantic classes via hierarchical and collective classification," in *Proceedings of the 23rd International Conference on Computational Linguistics*, ser. COLING '10. USA: Association for Computational Linguistics, 2010, p. 931–939.

[31] Y. Ma, E. Cambria, and S. Gao, "Label embedding for zero-shot fine-grained named entity typing," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 171–180. [Online]. Available: https://aclanthology.org/C16-1017

[32] Q. Ren, "Fine-grained entity typing with hierarchical inference," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, 2020, pp. 2552–2558.

[33] S. Shimaoka, P. Stenetorp, K. Inui, and S. Riedel, "Neural architectures for fine-grained entity type classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, 2017, pp. 1271–1280. [Online]. Available: https://aclanthology.org/E17-1119

[34] S. Murty, P. Verga, L. Vilnis, I. Radovanovic, and A. McCallum, "Hierarchical losses and new resources for fine-grained entity typing and linking," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 97–109. [Online]. Available: https://aclanthology.org/P18-1010

[35] J. Wu, R. Zhang, Y. Mao, H. Guo, and J. Huai, "Modeling noisy hierarchical types in fine-grained entity typing: A content-based weighting approach," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint

Conferences on Artificial Intelligence Organization, 7 2019, pp. 5264–5270. [Online]. Available: https://doi.org/10.24963/ijcai.2019/731

[36] T. Chen, Y. Chen, and B. Van Durme, "Hierarchical entity typing via multi-level learning to rank," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 8465–8475. [Online]. Available: https://aclanthology.org/2020.acl-main.749

[37] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, p. 809–816.

[38] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," ser. ICML'16. JMLR.org, 2016, p. 2071–2080.

[39] Y. Wang, X. Xin, and P. Guo, "An empirical study of pre-trained embedding on ultra-fine entity typing," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, pp. 2656–2661.

[40] H. Dai, Y. Song, and X. Li, "Exploiting semantic relations for fine-grained entity typing," in *Conference on Automated Knowledge Base Construction, AKBC 2020, Virtual, June 22-24, 2020*, D. Das, H. Hajishirzi, A. McCallum, and S. Singh, Eds., 2020. [Online]. Available: https://doi.org/10.24432/C5J017

[41] Q. Liu, H. Lin, X. Xiao, X. Han, L. Sun, and H. Wu, "Fine-grained entity typing via label reasoning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 4611–4622. [Online]. Available: https://aclanthology.org/2021.emnlp-main.378

[42] Y. Onoe, M. Boratko, A. McCallum, and G. Durrett, "Modeling fine-grained entity types with box embeddings," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 2051–2064. [Online]. Available: https://aclanthology.org/2021.acl-long.160

[43] H. Dai, Y. Song, and H. Wang, "Ultra-fine entity typing with weak supervision from a masked language model," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association

for Computational Linguistics, 2021, pp. 1790–1799. [Online]. Available: https://aclanthology.org/2021.acl-long.141

[44] F. Hou, R. Wang, and Y. Zhou, "Transfer learning for fine-grained entity typing," *Knowledge and Information Systems*, vol. 63, no. 4, pp. 845–866, Apr 2021. [Online]. Available: https://doi.org/10.1007/s10115-021-01549-5

[45] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "SpanBERT: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020. [Online]. Available: https://aclanthology.org/2020.tacl-1.5

[46] S. K. Pal and D. P. Mandal, "Fuzzy logic and approximate reasoning: An overview," *IETE Journal of Research*, vol. 37, no. 5-6, pp. 548–560, 1991.

[47] P. Hájek, "Basic fuzzy logic and bl-algebras," *Soft computing*, vol. 2, no. 3, pp. 124–128, 1998.

[48] V. Novák, I. Perfilieva, and J. Mockor, *Mathematical principles of fuzzy logic*. Springer Science & Business Media, 2012, vol. 517.

[49] E. van Krieken, E. Acar, and F. van Harmelen, "Analyzing differentiable fuzzy logic operators," *Artificial Intelligence*, vol. 302, p. 103602, 2022.

[50] A. d. Garcez and L. C. Lamb, "Neurosymbolic ai: the 3rd wave," *arXiv preprint arXiv:2012.05876*, 2020. [Online]. Available: https://arxiv.org/abs/2012.05876

[51] S. Badreddine, A. d'Avila Garcez, L. Serafini, and M. Spranger, "Logic tensor networks," *Artificial Intelligence*, vol. 303, p. 103649, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370221002009

[52] W. Cohen, F. Yang, and K. R. Mazaitis, "Tensorlog: A probabilistic database implemented using deep-learning infrastructure," *Journal of Artificial Intelligence Research*, vol. 67, pp. 285–325, 2020.

[53] W. W. Cohen, H. Sun, R. A. Hofer, and M. Siegler, "Scalable neural methods for reasoning with a symbolic knowledge base," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=BJlguT4YPr

[54] P. Minervini, S. Riedel, P. Stenetorp, E. Grefenstette, and T. Rocktäschel, "Learning reasoning strategies in end-to-end differentiable proving," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 6938–6949. [Online]. Available: http://proceedings.mlr.press/v119/minervini20a.html

[55] A. Campero, A. Pareja, T. Klinger, J. Tenenbaum, and S. Riedel, "Logical rule induction and theory learning using neural theorem proving," *arXiv preprint arXiv:1809.02193*, 2018. [Online]. Available: https://arxiv.org/abs/1809.02193

[56] G. Marra and O. Kuželka, "Neural markov logic networks," *arXiv preprint arXiv:1905.13462*, 2019. [Online]. Available: https://arxiv.org/abs/1905.13462

[57] M. Qu and J. Tang, "Probabilistic logic neural networks for reasoning," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 7710–7720. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/13e5ebb0fa112fe1b31a1067962d74a7-Abstract.html

[58] P. Hohenecker and T. Lukasiewicz, "Ontology reasoning with deep neural networks," *Journal of Artificial Intelligence Research*, vol. 68, pp. 503–540, 2020.

[59] T. Rocktäschel, "Deep prolog: End-to-end differentiable proving in knowledge bases," *AITP 2017*, p. 9, 2017.

[60] P. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter, "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 6545–6554. [Online]. Available: http://proceedings.mlr.press/v97/wang19e.html

[61] Z. Jiang and S. Luo, "Neural logic reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 3110–3119. [Online]. Available: http://proceedings.mlr.press/v97/jiang19a.html

[62] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[63] G. Sourek, V. Aschenbrenner, F. Zelezny, S. Schockaert, and O. Kuzelka, "Lifted relational neural networks: Efficient learning of latent relational structures," *Journal of Artificial Intelligence Research*, vol. 62, pp. 69–100, 2018.

[64] R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma *et al.*, "Logical neural

networks," *arXiv preprint arXiv:2006.13155*, 2020. [Online]. Available: https://arxiv.org/abs/2006.13155

[65] T. Li and V. Srikumar, "Augmenting neural networks with first-order logic," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 292–302. [Online]. Available: https://aclanthology.org/P19-1028

[66] G. Marra, M. Diligenti, F. Giannini, M. Gori, and M. Maggini, "Relational neural machines."

[67] G. Marra, F. Giannini, M. Diligenti, and M. Gori, "Integrating learning and reasoning with deep logic models," *arXiv preprint arXiv:1901.04195*, 2019. [Online]. Available: https://arxiv.org/abs/1901.04195

[68] ——, "Lyrics: A general interface layer to integrate logic inference and deep learning," *arXiv preprint arXiv:1903.07534*, 2019. [Online]. Available: https://arxiv.org/abs/1903.07534

[69] G. Marra, F. Giannini, M. Diligenti, M. Maggini, and M. Gori, "T-norms driven loss functions for machine learning," *arXiv preprint arXiv:1907.11468*, 2019. [Online]. Available: https://arxiv.org/abs/1907.11468

[70] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. T. Vechev, "DL2: training and querying neural networks with logic," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 1931–1941. [Online]. Available: http://proceedings.mlr.press/v97/fischer19a.html

[71] M. Diligenti, M. Gori, and C. Sacca, "Semantic-based regularization for learning and inference," *Artificial Intelligence*, vol. 244, pp. 143–165, 2017.

[72] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck, "A semantic loss function for deep learning with symbolic knowledge," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 5498–5507. [Online]. Available: http://proceedings.mlr.press/v80/xu18h.html

[73] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing, "Harnessing deep neural networks with logic rules," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016, pp. 2410–2420. [Online]. Available: https://aclanthology.org/P16-1228

[74] W. Dai, Q. Xu, Y. Yu, and Z. Zhou, "Bridging machine learning and logical reasoning by abductive learning," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 2811–2822. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/9c19a2aa1d84e04b0bd4bc888792bd1e-Abstract.html

[75] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. D. Raedt, "Deepproblog: Neural probabilistic logic programming," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 3753–3763. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html

[76] Q. Li, S. Huang, Y. Hong, Y. Chen, Y. N. Wu, and S. Zhu, "Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 5884–5894. [Online]. Available: http://proceedings.mlr.press/v119/li20f.html

[77] Q. Lu and L. Getoor, "Link-based classification," in *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, T. Fawcett and N. Mishra, Eds. AAAI Press, 2003, pp. 496–503. [Online]. Available: http://www.aaai.org/Library/ICML/2003/icml03-066.php

[78] P. Pavlidis and W. N. Grundy, "Combining microarray expression data and phylogenetic profiles to learn gene functional categories using support vector machines," 2000.

[79] K. Trohidis, G. Tsoumakas, G. Kalliris, I. P. Vlahavas *et al.*, "Multi-label classification of music into emotions." in *ISMIR*, vol. 8, 2008, pp. 325–330.

[80] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei, "Visual relationship detection with language priors," in *European conference on computer vision*. Springer, 2016, pp. 852–869.

[81] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1247–1250.