On the Capabilities of Logic Tensor Networks for Deductive Reasoning

Federico Bianchi,[1,2] Pascal Hitzler,[2]
[1]University of Milan-Bicocca
[2]Wright State University
federico.bianchi@disco.unimib.it, pascal.hitzler@wright.edu,

Abstract

Neural-symbolic integration is a field in which classical symbolic knowledge mechanisms are combined with neural networks. This is done to provide satisfactory computational capabilities from the network side and to exploit the descriptive power of symbolic reasoning. Logic Tensor Networks (LTNs) are a deep learning model that can be used to combine data with fuzzy logic to provide inferences and reasoning mechanisms over data. While LTNs have been shown effective in some contexts no detailed analysis on their capabilities for deductive logical reasoning has been conducted. In this paper we explore the capabilities and the limitations of LTNs in terms of deductive reasoning.

Introduction

Neural-symbolic learning and reasoning (Garcez, Lamb, and Gabbay 2008; Besold et al. 2017) involves integrating standard logical reasoning with neural networks with the aim of providing fast and robust computational methods for reasoning and explanation over data. Logic Tensor Networks (LTNs) are a deep learning model that comes from the neural-symbolic field: it integrates both logic and data in a neural network to provide support for neural symbolic-learning and reasoning (Serafini and Garcez 2016). LTNs use first-order fuzzy logic to express knowledge about the world: using fuzzy logic over classical first-order logic allows us to represent truth using continuous values in the interval $[0, 1]$ to represent the degree of truth.

Input to LTNs are data and axioms over (fuzzy) first-order predicate logic, e.g., parent(Ann, Susan), $\forall x, y : parent(x, y) \rightarrow ancestor(x, y)$. Two key components of logic tensor networks are the *grounding* of formulas and the *learning by best satisfiability*. With formula grounding we refer to the mapping of formulas to a vector space. For example, constants are mapped to n-dimensional vectors while function symbols are mapped to linear functions. A neural network can be used to compute the degree of truth of a given formula considering the embedded representation of constants and symbols.

Deep learning models (Goodfellow, Bengio, and Courville 2016) usually learn by optimizing a function; in LTNs this task is replaced with the task of best satisfiability: the model has to optimize the representation of each atom, function and predicate in such a way that the satisfiability of each formula is maximized. In this way the network learns the best possible parameters to represent both data and axioms.

The main advantages of LTNs are the following: i) it is possible to express knowledge using logical axioms over data ii) it is possible to tackle and solve standard machine learning tasks (e.g., classification) and iii) provide explanations using fuzzy logic over the trained network. Indeed, after training it is possible to make fuzzy inferences over data to obtain the degree of truth with respect to certain predicates. The model was tested with promising results on simple reasoning tasks (Serafini and Garcez 2016) and on semantic image interpretation (Donadello, Serafini, and d'Avila Garcez 2017).

An initial exploration of the reasoning capabilities of LTNs was done on the well-known smoker-friends-and-cancer dataset (Serafini and Garcez 2016). The dataset contains data about two groups of people for which friend relationships and smoking habits are given, while the fact of having cancer or not is only given for people in the first group. Axioms related to smoking properties (i.e., smoking implies cancer) are given to the network. The network learns to predict if people in the second group have cancer having learned the patterns present in the first group. More recently, LTNs were used on a semantic image interpretation task in which they learned to classify bounding boxes of images with the help of background knowledge (Donadello, Serafini, and d'Avila Garcez 2017). Still, an in-depth analysis of the deductive reasoning capabilities of LTNs remains to be done.

In this work we explore LTNs in the context of reasoning tasks, showing insights and properties of the model. We introduce two simple datasets that contain relationships and we define additional axioms over these datasets. These two datasets are used to evaluate deductive reasoning capabilities. We also perform some experiments on the computation time that is required to learn model parameters. Our results show that LTNs are a good model that can fit well the data and that is able to do simple deductive inferences.

The real added value of the model is that it lends itself to explanations, since it allows us to do after-training fuzzy inferences over the data. Nevertheless, the model generates some errors, in particular when multi-hop inferences are to be drawn, and thus some refinements over the general model might be required to improve the results.

The rest of the paper is organized as follows: in Section 2 we describe LTNs showing the basic definitions and the learning process, in Section 3 we introduce our experimental setting and we describe and evaluate the results of our experiments. Section 4 contains other related work. Finally, we end the paper in Section 5 with some conclusions and future work.

Logic Tensor Networks

LTNs use first-order fuzzy logic (Petr 1998) and embed atoms, functions, and predicates in a vector space. LTNs are inspired by Neural Tensor Networks (Socher et al. 2013) that have been shown to be effective in natural logic reasoning tasks (Bowman, Potts, and Manning 2015). In the following sections we will give a short primer on logic tensor networks and their learning methodology. More details on LTNs can be found in the paper in which they were first introduced (Serafini and Garcez 2016). To describe LTNs we will follow the definitions given by Serafini and Garcez.

Logic

LTNs are implemented over a logic called *Real Logic* that is described by a language L that contains a set of constants C, a set of function symbols F and a set of predicates P. In this language rules from fuzzy logic apply and connectives are interpreted as binary operations over real numbers in $[0, 1]$. For example *t-norms* are used in place of the conjunction from classical logic. The t-norm is an operation $[0, 1]^2 \rightarrow [0, 1]$ and different versions of the operation exist (Lukasiewicz, Gödel and product t-norms are some possible examples). Once the t-norm is chosen also the other connectives can be defined with respect to it. Thus, the use of t-norms and the other fuzzy connectives allows us to operate on real-values in the interval $[0, 1]$.

Grounding

Each element of the language L is grounded in the vector space. Constants are mapped to vectors in $\mathbb{R}^m$ while function symbols are mapped to functions in the vector space. An n-ary function symbol is mapped to an n-ary function $\mathbb{R}^{k \cdot n} \rightarrow \mathbb{R}^m$. Predicates are mapped to functions with co-domain in $[0, 1]$: $\mathbb{R}^{m*n} \rightarrow [0, 1]$; the predicate is mapped to a fuzzy subset that defines the degree of truth (membership to the set) for that predicate given its arguments.

Networks

The dimensionality of the vector of the constant is an hyper-parameter of the model. While constants are mapped to vectors, functions and predicates are mapped to actual operations over the vector space. We will use $\mathcal{G}(f)$ and $\mathcal{G}(P)$ to identify groundings of functions and predicates. Function symbols are implemented as linear functions: given f a symbol function of arity m and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^n$ are the groundings of m terms then the grounding for the symbol function f can be expressed as:

$$\mathcal{G}(f)(\mathbf{v}_1, \ldots, \mathbf{v}_m) = M_f \mathbf{v} + B_f \quad (1)$$

where $\mathbf{v} = \langle \mathbf{v}_1, \ldots, \mathbf{v}_m \rangle$, M_f is a transformation matrix and B_f is the bias. This operation can be encoded into a one-layer neural network.

Predicates are instead mapped to neural tensor operations (Socher et al. 2013), the output of the neural tensor network is given in input to a sigmoid such that the final output of the predicate is a value in the interval $[0, 1]$. The tensor operation is the following:

$$\mathcal{G}(P)(\mathbf{v}) = \sigma(u_P^T(tanh(\mathbf{v}^T W_P^{[1:k]} \mathbf{v} + V_P \mathbf{v} + B_P))) \quad (2)$$

σ is the sigmoid function while W, V, B and u are parameters to be learned by the network while k corresponds to the layer size of the tensor and is an hyper-parameter in the network.

Quantifiers like $\forall$ in fuzzy logic are defined with aggregation functions (like the *min*): this should consider an aggregation over an infinite number of instances, making it impossible to compute. Thus, quantifiers are implemented as aggregation operations over a subset of the domain space $\mathbb{R}^k$. Different possible implementations can be used to implement the aggregation for the universal quantifiers, for example *mean*, *min* and *hmean* (harmonic mean).

Learning to Satisfy Formulas

LTNs reduce the learning problem to a maximum satisfiability problem: the task is to find groundings for atoms, predicates and formulas that maximize the satisfiability of a given formula. For example, given the formula $parent(Susan, Ann)$, which describes the fact that *Susan* is one of *Ann*'s parents, the network will try to optimize the groundings of the predicate parent (i.e., the parameters in the tensor layer) and the groundings of $Susan$ and Ann (i.e., their respective two vectors) in such a way that the degree of truth of the formula is close to 1. Thus, the groundings are both the embedded representation of the atoms and the parameters in the networks that represent both functions and predicates; the values of these components can be learned through the use of back-propagation (Goodfellow, Bengio, and Courville 2016). The output of the learning process is a satisfiability score (in the interval $[0, 1]$) that can be considered similar to the value of the loss function in a standard deep learning setting.

We show an example of how grounding and the satisfiability are combined. For compactness, in this example we will identify the grounding of each element with a G as a superscript: given the formula $P(x, y) \wedge R(w, z)$, the groundings for the constants x, w, z, and y are retrieved (denoted with x^G). P and R are grounded to the respective operations: $P^G(x^G, y^G) \wedge R^G(w^G, z^G)$. The output of both predicates is a real value in [0,1] that can be aggregated with the use of the t-norm. LTNs will learn to optimize the groundings in such a way that the final value is close to 1 (i.e., the formula is satisfied).

Experiments

In this experimental section we aim to obtain answers for the following questions: i) what can LTNs learn and ii) how fast is the LTNs learning phase. To allow easy replication of our experiments we will first describe the datasets we use and then we will introduce some details on the general methodology we have followed during our experiments. Details that are related to a particular experiment will be given in the related section. For our experiment we use the original LTNs TensorFlow implementation[1] provided by the authors (Serafini and Garcez 2016). Datasets, code and results are available online with specific instructions on how to repeat our experiments[2]. We briefly summarize here the four experiments we ran:

- Experiments 1 and 2 will concentrate on a knowledge base completion task in which we will give to the network only true predicates and some axioms;
- Experiment 3 will compare LTNs with a simple deep learning baseline to provide insights about the strength and the limits of the model;
- Experiment 4 will show computational times related to experiments on learning with LTNs.

Definitions

By $\mathbf{KB}^S$ we denote an input (starting) knowledge base, and $\mathbf{KB}$ will denote the corresponding completed knowledge base (i.e., with all relevant logical consequences added). $\mathbf{KB}^T$ denotes the set of all inferences not in $\mathbf{KB}^S$, i.e., $\mathbf{KB}^T = \mathbf{KB} \setminus \mathbf{KB}^S$. In the experiment we will often show the performance over both $\mathbf{KB}$ and $\mathbf{KB}^T$ by putting results related to $\mathbf{KB}^T$ within parentheses.

Datasets

We use mainly two datasets for our experiments, the first one, called dataset **A**, represents a taxonomy that mainly contains hierarchies of classes (inspired by the DBpedia Ontology[3]). The taxonomy contains 25 nodes. Each node but one (the *root*) has an outgoing edge to its superclass (i.e., Cat is connected to Feline). Figure 1 shows the taxonomy used in the experiment.

The second dataset **P** is a parent-ancestor dataset that contains 17 nodes. Edges connect parents to one or more children, for a total of 22 parental relationships. Figure 2 shows the parental relationships.

We will test these two datasets on tasks in which we will have heavily unbalanced classes (more negative examples than positive ones). While our datasets are small compared to the ones currently used for knowledge base completion tasks (Bordes et al. 2013), we think that the results of our experiments can point out interesting capabilities of LTNs: can LTNs perform deductive reasoning over these simple datasets? Moreover, using these small datasets, results can be manually inspected to better understand where and how

[1] https://github.com/logictensornetworks

[2] https://github.com/vinid/ltns-experiments

[3] http://dbpedia.org

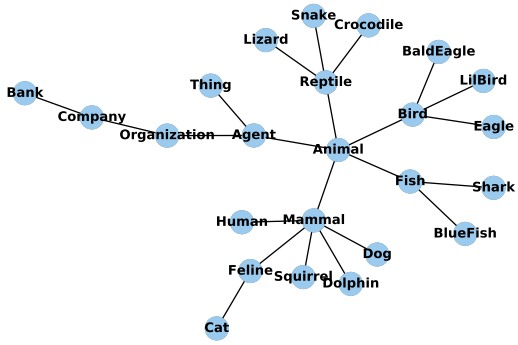

Figure 1: Taxonomy that represents the content of the **A** dataset

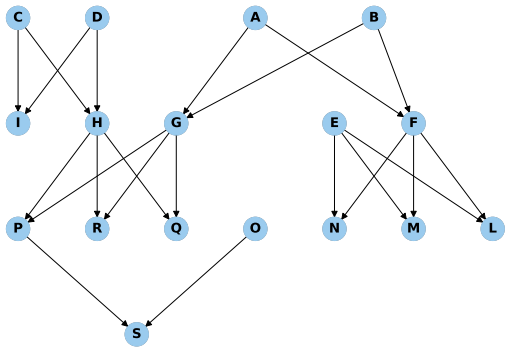

Figure 2: Representation of the parental relationships in the P dataset

the models fail to produce correct answers, a task that is more difficult with big knowledge bases.

Methodology

Given a dataset we define a set of axioms and we test a knowledge base completion task, showing hyper-parameter details. LTNs will receive in input data under the form of predicates (e.g., *parent(Ann,Susan)*) and axioms (e.g., $\forall x, y : parent(x, y) \rightarrow ancestor(x, y)$); the network will learn groundings for all the parameters and in the testing phase we will analyze predictions over data. Since different configurations of hyper-parameters are possible we run multiple models and we re-run each model multiple times (to check variations due to random initialization). After a first phase of trial and error we set as static the following parameters: optimizer *RMSprop*, bias $-1e^{-5}$, learning rate 0.01, decay 0.9. We tested three different aggregation functions for the universal quantifiers (harmonic mean, mean and min), two tensor layer sizes (10 and 20) for the tensor network and two embedding sizes for constants (10 and 20 dimensions).

Evaluation Measures To evaluate the models we use the Mean Absolute Error (MAE), Matthews correlation coefficient (Matthews 1975) that is often regarded as stable when classes are unbalanced, F1 score, precision, and recall. When we compute MAE we will compute the absolute distance between the fuzzy predictions and the actual true values; this will give us the possibility of understanding how good are models with a continuous error value. When computing the other measures we will round the scores to the nearest integer in such a way that we compare only binary scores. We consider prediction values higher than 0.5 as 1 and vice-versa. While this is a strong approximation over the degree of truth given by fuzzy logic it is still useful to understand the performance of the model. We will also report accuracy to summarize the performance of the model when necessary. In general, we select the best model for each experiment by considering the one with the highest F1 score.

Experiment 1: Taxonomy Reasoning

For the A dataset we ask the LTNs to learn the following axioms:

- $\forall a, b, c \in A : (sub(a, b) \wedge sub(b, c)) \rightarrow sub(a, c)$
- $\forall a \in A : \neg sub(a, a)$
- $\forall a, b : sub(a, b) \rightarrow \neg sub(b, a)$

Where sub identifies the subclass relation in the dataset (e.g., $sub(Cat, Feline)$). The objective of this experiment is to see if LTNs can generate the transitive closure starting from a dataset using the axioms. Data contained in the A dataset is our $\mathbf{KB}^S$ while the edges needed to generate the transitive closure will be our $\mathbf{KB}^T$. We compare the predictions of LTNs (computed as the prediction over $sub(x, y)$ given x, y) with the actual transitive closure of the graph. We recall that $\mathbf{KB}^S$ contains only true predicates (e.g., $sub(Cat, Feline)$) while we ask the model to perform inferences also over predicates that are false (e.g., we evaluate $sub(Feline, Cat)$ expecting a value close to 0).

Table 1 shows results for the knowledge completion tasks of the top performing model and one of the worst performing ones: the top performing model had a satisfiability equal to 0.99 while one of the worst ones had a satisfiability of 0.56. The top-performing model was initialized with a layer size in the tensor network of 20 and a dimension of the embeddings equal to 20; the best universal aggregator was the mean aggregator.

The best model over $\mathbf{KB}$ is able to fit well the data since the F1 measure show good performance over the entire knowledge base (F1 = 0.64). LTNs are prone to generate false positives: the model generates 36 false positives with respect to 55 true positives and 26 false negatives with respect to 459 true negatives.

The performance drops when we consider only $\mathbf{KB}^T$ elements for testing (F1 = 0.51), this means that LTNs are in this case not able to capture some more complicated inferences.

Still, the approach is better than a binary random baseline. The accuracy of the model with the best satisfiability is 0.89, while a naive classifier that predicts only zeros would have reached an accuracy equal to 0.85. This is important to remark since the two classes are ill-balanced.

Qualitative Analysis Analyzing the prediction of LTNs we found that in some cases the model correctly predicts multi-hop logical inferences (e.g., sub(Cat, Animal) close to 1), but fails on other simple inferences (e.g., sub(Cat, Bird) close to 1). When there is not enough information regarding the relationship between two elements (e.g., Cat and Bird) the model has difficulties to predict the correct answer.

Summary of the outcomes

- LTNs fit the data well;
- Multi-hop inferences tend to be more difficult;
- As expected performance increases with satisfiability.

Experiment 2: Ancestors Reasoning

For the P dataset we train LTNs with the following axioms:

- $\forall a, b \in P : parent(a, b) \rightarrow ancestor(a, b)$
- $\forall a, b, c \in P : (ancestor(a, b) \wedge parent(b, c)) \rightarrow ancestor(a, c)$
- $\forall a \in P : \neg parent(a, a)$
- $\forall a \in P : \neg ancestor(a, a)$
- $\forall a, b \in P : parent(a, b) \rightarrow \neg parent(b, a)$
- $\forall a, b \in P : ancestor(a, b) \rightarrow \neg ancestor(b, a)$

Thus, we combine the knowledge of these axioms with the data of the parental relationships. We distinguish two different relationships in this dataset *parent* (i.e., $parent(x, y)$ means that x is a parent of y) and *ancestor* (i.e., $ancestor(x, y)$ means that x is a ancestor of y).

$\mathbf{KB}^S$ contains only the parental relationships shown in Figure 2 (e.g., $parent(C, I)$). The task we will test is to infer the complete knowledge base for the *ancestor* predicate, to which we will refer to as $\mathbf{KB}_a$; therefore, we would like LTNs to learn if an ancestor relationship is true or false for two given nodes only from axioms and parental data. The representation for the $ancestor$ predicate should be generated from the knowledge in the axioms since no data about it is provided.

We will also test how the model performs over the set of ancestor formulas that require multi-hop inferences to be inferred (i.e., those that cannot be directly inferred from $\forall a, b \in P : parent(a, b) \rightarrow ancestor(a, b)$), we will refer to this as $\mathbf{KB}_a^T$: those ancestors pairs for which the parent pair is false (e.g., $ancestor(C, S)$). As before, we recall that $\mathbf{KB}^p$ contains only true predicates (e.g., $parent(C, I)$) while we ask the model to perform inferences over the ancestor dataset ($\mathbf{KB}_a$) that also contains predicates that should be inferred as false (e.g., $ancestor(I, C)$).

We do this to understand if LTNs are able to pass the information from the $parent$ predicate to the $ancestor$ predicate and if this is enough to give to the network the possibility of making even more complex inferences that are related to chains of ancestors.

The best performing model for this task (with hmean, 10 dimensional embeddings, 10 neural tensor layers) over $\mathbf{KB}_a$

Table 1: Performance measures on the A dataset. Value out of the parentheses are computed over the complete $\mathbf{KB}$ while those within parentheses are computed only on the part of the $\mathbf{KB}$ that was not in the initial set of data.

Satisfiability	MAE	Matthews	F1	Precision	Recall
0.99	**0.12** (0.12)	**0.58** (0.45)	**0.64** (0.51)	**0.60** (0.47)	0.68 (0.55)
0.56	0.51 (0.52)	0.09 (0.06)	0.27 (0.20)	0.20 (0.11)	**0.95** (0.93)
Random	0.50 (0.50)	0.00 (0.00)	0.22 (0.17)	0.14 (0.10)	0.50 (0.50)

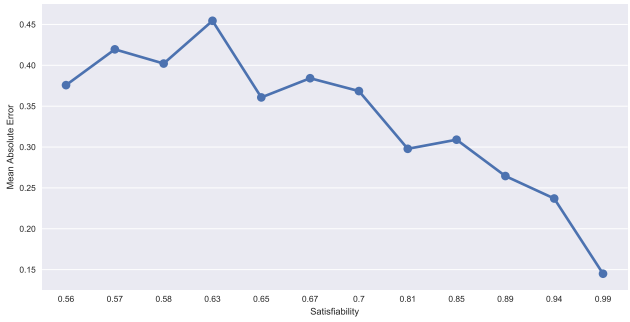

Figure 3: Average MAE for the ancestors tasks on rounded level of satisfiability. MAE decreases with the increase of satisfiability.

had an F1 score of 0.77. If we do not consider the ancestor predicates that can be directly inferred from the axioms ($\mathbf{KB}_a^T$), the model correctly infers 22 ancestors while generating 25 false positives: the F1 is equal to 0.62. Again, the network seems to be able to fit the data quite well, but it still generates errors on multi-hop inferences.

As another experiment over satisfiability, in Figure 3 we show the relation between the MAE computed on $\mathbf{KB}_a$ and the level of satisfiability. To draw this figure we run multiple experiments with LTNs and computed the mean MAE by aggregating the satisfiability levels rounded to 2 decimal digit. It is clear that the error decreases with the increase of the satisfiability level and thus LTNs are able to learn and infer some knowledge. This proves again that the model is able to learn the originally not known ancestor relationships from the combination of data and rules.

Comparison With Added Axioms To provide a better understanding of this experiment we decided to add two axioms to the previous set. These two axioms explicitly state the relationships between parents and ancestors:

- $\forall a, b, c \in P : (ancestor(a, b) \wedge ancestor(b, c)) \rightarrow ancestor(a, c)$
- $\forall a, b, c \in P : (parent(a, b) \wedge parent(b, c)) \rightarrow ancestor(a, c)$

Table 2 shows the comparison between the approach without the new axioms (*Six Axioms*) and with the new axioms (*Eight Axioms*) on the ancestor dataset. Performances were computed on the two models with the highest satisfiability (both around 0.99). The top-performing models for both *Six Axioms* and *Eight Axioms* were initialized with a layer size in the tensor network of 10 and a dimension of the embeddings equal to 10; the best universal aggregator was the hmean aggregator. Results show that the new axioms are beneficial for the network, that is actually able to learn well the relationships. Still, the precision over $\mathbf{KB}_a^T$ has increased by 0.19 points (the difference between the results within parentheses).

One interesting result about this is related to the fact that the network is able to learn a good representation for the $ancestor$ predicate just from the axioms.

Qualitative Analysis LTNs allows us to do fuzzy inferences after training. The model is able to answer queries on fuzzy formulas that were not in the original training data. For example, $\forall a, b : ancestor(a, b) \rightarrow \neg parent(b, a)$ has generally a value close to 1 in our experiments.

Summary of the outcomes

- Satisfiability is strongly related with performance of the model: the higher the satisfiability the lower the error;
- LTNs learn to pass information quite efficiently (information on $parent(x, y)$ is passed to $ancestor(x, y)$). Still, some more complicated inferences are difficult;
- More axioms increase the performance of the model.

Experiment 3: Comparison with a Multi-Input Network

In this experiment, we want to compare LTNs with a simple deep learning architecture on a common task. Starting from the complete knowledge base of parents and ancestors we randomly divide data into the training set and test set. Training data consists of 100 parent predicates (both true and false) and 100 ancestor predicates[4] (both true and false); test set contains 189 parent predicates and 189 ancestor predicates[5]. We thus tackle this problem by considering a classification setting that can be solved with the use of deep learning models.

We built a simple multi-input architecture that took as input three one-hot encoded representations of the pairs of atoms and the predicates (e.g., *Susan*, *Ann*, *parent*). This is not the most optimized architecture to solve this task, but it is useful to understand the performance of LTNs compared with classical deep learning approaches. We trained the network using binary cross-entropy and the RMSprop gradient optimization algorithm over 5,000 epochs with a 20% validation split. To reduce possible effects of overfitting we use

[4]Note that the training set contains very few examples that are positive

[5]We tested different random subsets of training and testing, but the results tend to be similar

Table 2: Ancestor completion task with different number of axioms. Value out of the parentheses are computed over the complete $\mathbf{KB}_a$ while those within parentheses are computed on $\mathbf{KB}_a^T$.

Type	MAE	Matthews	F1	Precision	Recall
Six Axioms	0.16 (0.17)	0.73 (0.61)	0.77 (0.62)	0.64 (0.47)	**0.96** (0.92)
Eight Axioms	**0.14** (0.14)	**0.83** (0.69)	**0.85** (0.72)	**0.80** (0.66)	0.89 (0.79)

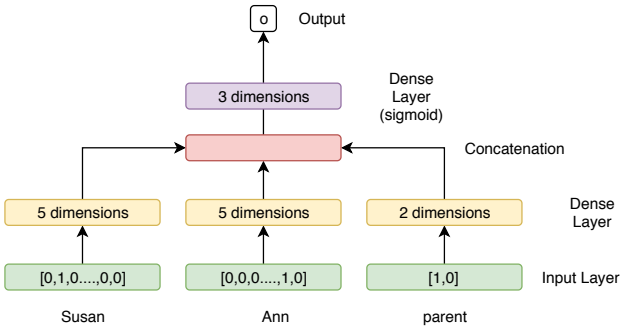

Figure 4: Baseline Multi-input architecture

L2 regularization (we experimentally found that results were better with it than without). We show this architecture in Figure 4 where we also show the dimensions of the layers.

The network is trained to detect if a predicate, given two constants, is true or false (binary outcome). LTNs are implicitly trained on the same task: we train the network over best satisfiability given the data in input and the six axioms used in the previous setting.

The performance of the models is computed over the 189 ancestor test predicates. We ignore the *parent* predicates in this setting because there is little to no knowledge about how to predict if a parental relationship in the test set is true or false from the dataset.

Results show that the multi-input network achieves an accuracy equal to 0.84 while accuracy for LTNs was around 0.89; while the accuracies are comparable an in-depth analysis with other measures revealed that the recall for the multi-input was 1 and its precision was 0.12, while LTNs had a lower recall (0.66) but a much higher precision (0.57). A naive model that predicts only zeros (since classes are unbalanced) would have reached an accuracy equal to 0.84. The multi-input architecture tends to overfit in this task in which most of the classes are 0. It is anyway important to note that it is difficult for the multi-input architecture to *understand* the task, while LTNs are helped by the axioms.

However, the results show that while LTNs are good for learning logical rules, their accuracy is still comparable to the one obtained by neural-networks. Moreover, the multi-input architecture would require more control on overfitting, while the logical axioms used in LTNs seems to provide a natural way to define some constraints over the vector space and to reduce possible overfitting. Nevertheless, different deep learning architectures with a different set of parameters might generate better results.

Using classical neural networks we lose the ability to define high-level semantics to the data. For example, LTNs can be used in combination with quantifiers to make inferences over data using new axioms on which the network was not trained (e.g., $\forall x, y : parent(x, y) \rightarrow \neg ancestor(y, x)$ has a high truth value).

As shown in the recent work on LTNs on semantic image interpretation one key element of success might be the use of LTNs over deep learning architectures (Donadello, Serafini, and d'Avila Garcez 2017); this would allow augmenting data with semantic information that will make it possible to explain predictions.

Summary of the outcomes

- Results show that performance on this simple task is comparable to a naive network;
- Axioms in LTNs seem to provide a useful way of defining constraints over the space of the solutions that might reduce the possibility of overfitting;
- The main advantage of LTNs resides in the possibility of making inferences after training.

Experiment 4: Time to Learn

In this last experiment, we investigate how fast LTNs are in the learning context. We consider the following experimental setting: we generate a range of N constant and N predicates and we evaluate different combinations of them. We divide this experiment in three by considering unary, binary and ternary predicates of the following from $\forall x : pred_n(x)$, $\forall x, y : pred_n(x, y)$, $\forall x, y, z : pred_n(x, y, z)$, we therefore test only predicates that are universally quantified. We compute 5,000 training epochs to learn the parameters of 4, 8, 12, 20, 30 constants with 4, 8, 12, 20, 30 (universally quantified) predicates of arity one, two and three: this means that in the setting with 4 constants and 8 predicates of arity 3 we introduce 4 constants (a, b, c, d) in the model and 8 predicates $(pred_1, pred_2, \ldots, pred_8)$ and each predicate is universally quantified (e.g., $\forall x, y, z : pred_1(x, y, z)$). Size of the embedded representation in this experiment is 10. Experiments were run using a compiled version of Tensorflow on an i7 machine.

Analysis Figures 5, 6, 7 show the seconds needed to complete the learning phase for each setting. While it is clear that constants have an influence on computational time (since they are training data) we can also state that predicates and their arity have a notable computational impact upon the learning phase. With a low number of constants and predicates (e.g., 4) the training time is not much different in all the settings, but as soon as the number of constants increases the model requires more time to learn. The arity of the predicates seem to be the element with the higher impact on the learning time: this is an expected result since the universal

quantifier has to cover multiple elements in the ternary case. Since experiments were run on a CPU we expect training time to be shorter on GPU [6].

Summary of the outcomes

- Time to learn the parameters is highly influenced by the arity of the predicates;

Other Experimental Notes

In this section, we briefly describe other experimental results that are interesting for the community. While the following assertions are derived from empirical experiments they might still be useful for the reader who wants to start using LTNs.

LTNs as all deep learning model suffers from optimization problems: in our experiments we often found the model reaching local minima. Global optimization tools might help in a better parameter optimization search.

In our experiments LTNs often predicted the class Cat to be a subclass of the class Bird. This error might be due to missing knowledge in the $\mathbf{KB}$. The network is not able to understand the difference between the two since they come from different branches of the taxonomy. In general, it seems that LTNs predict many false positives, while they are better in detecting true negatives. This seems due to the fact that true negatives in our experiments can be directly inferred from the axioms: for example, $\forall a : \neg ancestor(a, a)$ gives a good amount of information to the model about the fact that each constant occurring in both parameters of the predicate $ancestor$ should generate negative values.

If the model fits the data too well (i.e., it overfits) the performance over the test set decreases. While this is a common event for machine learning models and there are techniques to prevent this, applying these to LTNs is not so straightforward: cross-validation would require us to provide completeness information to the training set, that would bias the reasoning task.

We tested different sets of hyper-parameters and we release results on the tested tasks online. While this was not the primary scope of the paper it is still important to estimate the effects of the hyper-parameters to fully evaluate the approach. Nevertheless, we empirically find out that increasing the layers of the tensor network and the size of the embeddings makes the model much more difficult to optimize.

After paper acceptance a new version of LTNs was released by the original authors: this last version is easier to optimize and shows a slight increase in performance over the F1 measure.

Related Work

In this section we summarize some related approaches that have been introduced in the state of the art. We refer to Garcez, Lamb, and Gabbay; Besold et al. for discussions of different neural-symbolic approaches proposed in literature: in this section we will only discuss a few of these approaches and we will also describe some related methods.

One of the main points of discussion that has involved the artificial community in the last decades is the relationship between symbolic artificial intelligence and connectivist (i.e., related to neural networks) artificial intelligence (Minsky 1991). In recent years deep learning approaches have shown great computational capabilities (Goodfellow, Bengio, and Courville 2016), but still these approaches do not achieve the same reasoning and knowledge transformation abilities that symbolic approaches show. On the other hand, symbolic artificial intelligence suffers from computational limits and the knowledge acquisition bottleneck, i.e., the need to generate high-quality knowledge bases, which is usually done manually. A different voice in this group comes from the neural-symbolic field, where the task is to bring together the two worlds of symbolic artificial intelligence and neural networks (Garcez, Gabbay, and Broda 2002; Hammer and Hitzler 2007; Garcez, Lamb, and Gabbay 2008; Garcez et al. 2015).

In the current work we have explored only LTNs, but there are different approaches in the field that have been introduced. One of the most famous approaches of neural-symbolic integration are the Knowledge Based Artificial Neural Networks (KBANNs) (Towell and Shavlik 1994). KBANNs where one of the first approaches to integrate propositional clauses with data, developed at the same time as the closely related propositional core method (Hölldobler and Kalinke 1994). Lifting these results towards first-order logic, however, has been proven difficult and limited to toy-size knowledge bases (Hitzler, Hölldobler, and Seda 2004; Gust, Kühnberger, and Geibel 2007; Bader, Hitzler, and Hölldobler 2008).

On the other hand, there are other approaches from the Statistical Relational Learning field that do not integrate neural networks with logic, but tackle the problem in a symbolic manner by also combining statistical information. Examples of this category are ProbLog (De Raedt and Kimmig 2015) that is an example of probabilistic logic programming language and Markov Logic Networks (MLNs) are a statistical relational learning model that has been shown to be effective on a large variety of tasks (Richardson and Domingos 2006; Meza-Ruiz and Riedel 2009). The intuition behind MLNs and LTNs is similar since they both base their approach on logical languages. MLNs defines weights for formulas and interpret the world by considering it under a probabilistic point of view while LTNs use fuzzy logic combined with a neural architecture to generate their inferences.

Conclusions and Future Work

Conclusions

LTNs can be shown to obtain good results on reasoning tasks when optimal satisfiability conditions are met. This is often difficult to reach and using the model with a low degree of satisfiability can generate bad inferences. Nevertheless, LTNs show interesting capabilities and their ability to mix logic and data might prove to be a valuable resource. LTNs

[6] to show an effective comparison between different predicates we decided to show results computed with a CPU: with the GPU it was more difficult to highlight the differences between these experiments

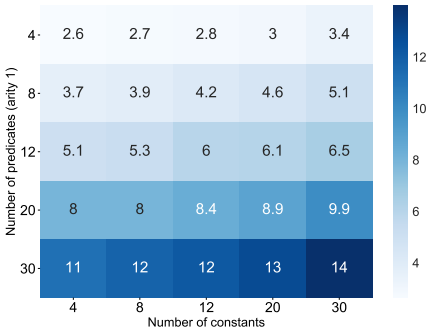

Figure 5: Computational times in seconds for predicates of arity one and constants

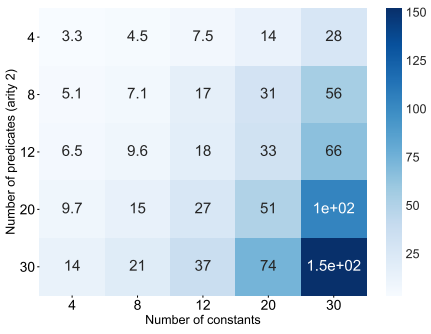

Figure 6: Computational times in seconds for predicates of arity two and constants

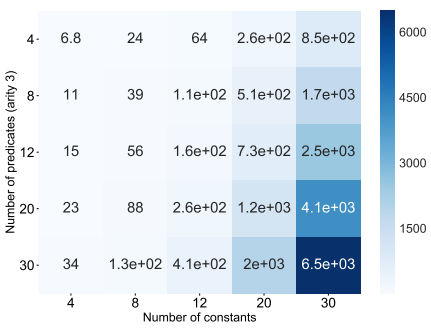

Figure 7: Computational times in seconds for predicates of arity three and constants

fit well the data and can be used to make some simple inferences. More complex inferences (multi-hop) are more difficult to capture in the model.

The main problem encountered in our experiments is related to erroneous prediction generated by the LTNs and scalability issues. We think that the former problem might be solved with a more accurate use of logic constraints: for example, in the *ancestor* experiment, adding notions about the concepts of "siblings" might help the network to perform better. While a more efficient use of computational resources could help in reducing the latter problem we encountered.

Future Work

While results have shown that LTNs are able to capture logic semantics in the vector space, they should also be compared with other statistical relational learning methods like MLNs on similar tasks.

Another possible next step is to apply LTNs on bigger knowledge bases defined in the state of the art (Bordes et al. 2013). We expect the ability to make fuzzy inferences over the trained model to be of great help in link prediction tasks over knowledge bases.

An interesting development of this work could be evaluating the generated groundings: constants in LTNs have an associated vector and thus it is possible to compute the similarity in the vector space between constants. This might be interesting in the context of knowledge graph embeddings (Bordes et al. 2013): vector representations of entities and relationships of a knowledge graph.

Acknowledgment

We thank Luciano Serafini and Artur d'Avila Garcez for their comments and suggestions. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

Bader, S.; Hitzler, P.; and Hölldobler, S. 2008. Connectionist model generation: A first-order approach. *Neurocomputing* 71(13-15):2420–2432.

Besold, T. R.; d'Avila Garcez, A. S.; Bader, S.; Bowman, H.; Domingos, P. M.; Hitzler, P.; Kühnberger, K.; Lamb, L. C.; Lowd, D.; Lima, P. M. V.; de Penning, L.; Pinkas, G.; Poon, H.; and Zaverucha, G. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR* abs/1711.03902.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, 2787–2795.

Bowman, S. R.; Potts, C.; and Manning, C. D. 2015. Learning distributed word representations for natural logic reasoning. In *Proceedings of the Association for the Advancement of Artificial Intelligence Spring Symposium (AAAI)*, 10–13.

De Raedt, L., and Kimmig, A. 2015. Probabilistic (logic) programming concepts. *Machine Learning* 100(1):5–47.

Donadello, I.; Serafini, L.; and d'Avila Garcez, A. 2017. Logic tensor networks for semantic image interpretation. In *IJCAI*, 1596–1602.

Garcez, A.; Besold, T. R.; De Raedt, L.; Földiak, P.; Hitzler, P.; Icard, T.; Kühnberger, K.-U.; Lamb, L. C.; Miikkulainen, R.; and Silver, D. L. 2015. Neural-symbolic learning and reasoning: contributions and challenges. In *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches, Stanford*.

Garcez, A. S. d.; Gabbay, D. M.; and Broda, K. B. 2002. *Neural-Symbolic Learning System: Foundations and Applications*. Berlin, Heidelberg: Springer-Verlag.

Garcez, A. S.; Lamb, L. C.; and Gabbay, D. M. 2008. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Gust, H.; Kühnberger, K.; and Geibel, P. 2007. Learning models of predicate logical theories with neural networks based on topos theory. In Hammer, B., and Hitzler, P., eds., *Perspectives of Neural-Symbolic Integration*, volume 77 of *Studies in Computational Intelligence*. Springer. 233–264.

Hammer, B., and Hitzler, P., eds. 2007. *Perspectives of Neural-Symbolic Integration*, volume 77 of *Studies in Computational Intelligence*. Springer.

Hitzler, P.; Hölldobler, S.; and Seda, A. K. 2004. Logic programs and connectionist networks. *J. Applied Logic* 2(3):245–272.

Hölldobler, S., and Kalinke, Y. 1994. Ein massiv paralleles modell für die logikprogrammierung. In *WLP*, 89–92.

Matthews, B. W. 1975. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405(2):442–451.

Meza-Ruiz, I., and Riedel, S. 2009. Jointly identifying predicates, arguments and senses using markov logic. In *NAACL*, 155–163. Association for Computational Linguistics.

Minsky, M. L. 1991. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI magazine* 12(2):34.

Petr, H. 1998. Metamathematics of fuzzy logic, vol. 4 of trends in logicstudia logica library.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine learning* 62(1-2):107–136.

Serafini, L., and Garcez, A. S. d. 2016. Learning and reasoning with logic tensor networks. In *Conference of the Italian Association for Artificial Intelligence*, 334–348. Springer.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, 926–934.

Towell, G. G., and Shavlik, J. W. 1994. Knowledge-based artificial neural networks. *Artificial intelligence* 70(1-2):119–165.