Investigation on Logical Tensor Networks for Multi-Class Single Label Classification

Zhen Wu
zhw87@pitt.edu
Undergraduate Student
School of Computing and Information, University of Pittsburgh

ABSTRACT

The fast learning ability of Neural Networks and good interpretability of Symbolic Systems called for a combination of them to take advantage of both sides. Thus, Neural-Symbolic (NS) Systems were born to bridge the gap. This paper investigates one kind of NS system, Logical Tensor Networks (LTN) which are defined on Real Logic, on a multi-class single-label classification task as a comparison to classical classifiers. We have shown that LTN outperforms other classifiers and has no overfitting problems which confirms the good ability of LTN in classification tasks.

KEYWORDS

Neuro-Symbolic, Logical Tensor Networks, Classification

1 INTRODUCTION

1.1 Background

As humans, we process strong reasoning and problem-solving abilities. For a long time, researchers are trying to build intelligent systems that can mimic human brains[6]. The current implementations of neural networks have shown their ability of fast learning and processing unstructured data, such as images, audio, and texts[9]. However, despite their learning ability, neural networks require a large amount of labeled data and lack interpretability[9], that is, they predict "what the result is" instead of "why this is the result." This is where the symbolic system sheds light on. Symbolic systems are good at logical reasoning and interpretability to obtain explainability[10]. Based on these, the combination of neural networks and symbolic systems, Neuro-Symbolic (NS) system, integrates the strengths and bridge the gap between the two systems to achieve effective and interpretable learning and reasoning. Currently, NS systems have been applied to areas including image recognition and detecting relationships between objects (ex. a person is "riding" a horse), knowledge learning, question-answering and reinforcement learning, and have shown effective outcomes[9].

1.2 Logical Tensor Networks

The Logical Tensor Networks (LTN) are one exploration of NS systems that perform well in learning and reasoning with rich knowledge[1]. They are implemented based on Real Logic which is defined on the $\mathcal{L}$ language formula that allows us to specify the relationship between variables. $\mathcal{L}$ consists of a set of constants (objects) $\mathcal{C}$, functional symbols $\mathcal{F}$, and relational symbols $\mathcal{P}$ (predicates). And each object belongs to a domain $\mathcal{D}$. For example, the $\mathcal{L}$-language formula expression $\forall x \forall y(is\_friend(x, y) \rightarrow is\_friend(y, x))$ implies a symmetric logic relation of friendship of the function *is_friend* in the domain of all living people[1].

In addition, Real Logic symbols are grounded into real-value features: domains are interpreted as tensors of the real fields $\mathfrak{R}$, objects are tensors of real values, functions are tensor operations, and predicates are treated as tensor operations that projects real values to the interval [0,1][1]. To emphasize the real value projection in LTN, the term *grounding* ($\mathcal{G}$) is introduced to assign each symbol in the $\mathcal{L}$ a sequence of tensors such that the following mappings are satisfied[1]:

- a constant symbol → a tensor in its domain
- a function symbol → tensors from input domain to output domain
- a predicate symbol → tensors from input domain to [0,1]

2 RELATED WORK

First proposed in 2016[8], LTN has been applied to multiple areas. In [7], LTN was examined in semantic image interpretation for object detection and part-of relation between them, where the results outperformed other state-of-the-art object detectors. [2] implemented LTN by injecting prior knowledge on objects and their semantics into the architecture of reinforcement learning agents. In [3], LTN's capabilities and limitations in deductive reasoning were evaluated, showing that LTN has good ability on reasoning and simple inference tasks.

The new version of LTN was implemented using PyTorch[4] called LTNtorch. LTNtorch has been applied to multiple tasks such as regression, clustering, multi- or binary- classifications, pattern recognition, and shown good performance. The source code and libraries are available on [5]. This paper will investigate the performance of LTNtorch on multi-class single-label data as a comparison to classical classification methods, such as Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Decision Tree.

3 METHODOLOGY

3.1 Dataset

This paper will apply LTNtorch model on mobile phone features dataset (from Kaggle dataset: https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification). The dataset contains 20 mobile phone features such as battery power, clock speed, Wi-Fi, and internal memory. The target label we want to predict has 4 possible values (0,1,2,3) indicating the range or level of the mobile phones' price based on the given features. Larger class number means higher price. There are 2000 data samples in total, and each data sample only belongs to one single label. The data samples are split into training and testing sets where the training dataset has 1400 samples and the testing dataset has 600 samples. We use DataLoader from PyTorch for both train and test data.

3.2 LTN model

3.2.1 *Basic definitions.*

Domain $\mathcal{D}$.

- *items*, denoting the data samples from the mobile phone dataset;
- *labels*, denoting the phone price class labels.

Variables.

- x_0,x_1,x_2,x_3 for the data samples of price class 0,1,2,3, respectively;
- x for all data samples;
- $\mathcal{D}(x_0) = \mathcal{D}(x_1) = \mathcal{D}(x_2) = \mathcal{D}(x_3) = items$, meaning that the domain of all data samples is *items*.

Constants.

- l_0,l_1,l_2,l_3 for the 4 price classes 0,1,2,3, respectively;
- $\mathcal{D}(l_0) = \mathcal{D}(l_1) = \mathcal{D}(l_2) = \mathcal{D}(l_3) = labels$, meaning that the domain of all price labels is *labels*.

Predicates.

- $\mathcal{P}(x,l)$, denoting that data sample x is classified as class l;
- $\mathcal{D}_{in}(\mathcal{P}) = items, labels$, meaning that the input domain of the predicate $\mathcal{P}$ are *items* (domain of x) and *labels* (domain of l).

Axioms.

- $\forall x_0\ \mathcal{P}(x_0,l_0)$: all samples of class 0 should have label l_0;
- $\forall x_1\ \mathcal{P}(x_1,l_1)$: all samples of class 1 should have label l_1;
- $\forall x_2\ \mathcal{P}(x_2,l_2)$: all samples of class 2 should have label l_2;
- $\forall x_3\ \mathcal{P}(x_3,l_3)$: all samples of class 3 should have label l_3.

Grounding.

- $\mathcal{G}(items) = \mathfrak{R}^{20}$, denoting that items are described by 20 features;
- $\mathcal{G}(labels) = \mathbb{N}^4$, we use one-hot encoding for each of the 4 classes;
- $\mathcal{G}(x_0) \in \mathfrak{R}^{m_0 \times 20}$, meaning that $\mathcal{G}(x_0)$ is a sequence of m_0 samples of class 0;
- $\mathcal{G}(x_1) \in \mathfrak{R}^{m_1 \times 20}$, meaning that $\mathcal{G}(x_1)$ is a sequence of m_1 samples of class 1;
- $\mathcal{G}(x_2) \in \mathfrak{R}^{m_2 \times 20}$, meaning that $\mathcal{G}(x_2)$ is a sequence of m_2 samples of class 2;
- $\mathcal{G}(x_3) \in \mathfrak{R}^{m_3 \times 20}$, meaning that $\mathcal{G}(x_3)$ is a sequence of m_3 samples of class 3;
- $\mathcal{G}(x) \in \mathfrak{R}^{(m_0+m_1+m_2+m_3) \times 20}$, meaning $\mathcal{G}(x_0)$ is a sequence of all the samples;
- $\mathcal{G}(l_0)$=[1,0,0,0], $\mathcal{G}(l_1)$=[0,1,0,0], $\mathcal{G}(l_2)$=[0,0,1,0], $\mathcal{G}(l_3)$=[0,0,0,1];
- $\mathcal{G}(\mathcal{P}|\ \theta)$: $x,l \mapsto l^T \cdot softmax(\text{MLP}_\theta(x))$, where softmax will give us probabilities between [0,1] and MLP has 4 output neurons corresponding to 4 possible class labels. Multiplying MLP's output with the one-hot vector encoding l returns the probability of the class denoted by l.

3.2.2 *Model Implementation.*

Metrics. Two metrics to evaluate the model include classification accuracy and satisfiability, or satisfaction level, which measures how well our model satisfies the knowledge base we defined previously. Our goal is to maximize the satisfaction level in training.

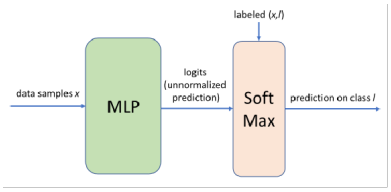

Figure 1: Two components of the basic model: an MLP for logits and softmax for class probabilities.

Basic Models. The basic models include two parts: 1) an MLP outputting logits (un-normalized probabilities) for the four classes; 2) a softmax layer taking a labeled pair of (x,l) and logits from the first model to calculate the probabilities for predicting class l (shown in Figure 1).

The two parts are separated such that we can calculate classification accuracy and satisfaction level individually. The logits computed from MLP are used to compute the accuracy score. For each data sample within the data loader, the class label prediction is the one with the highest logits value among the four classes. Then the overall accuracy score of the data loader is computed using *accuracy_score* metrics from scikit-learn. Whereas the satisfaction level is calculated via a mean aggregator. Within each data loader, data is divided into four groups by the four class labels. Then for each group, we pass the data samples and class label to the softmax layer and output a probability of the sample being classified as the given class. The goal is to see how well the current model prediction satisfies the axioms defined previously. The mean aggregator takes probabilities from each of the four groups and output an aggregated probability. The overall satisfaction level of the data loader is the mean of all aggregated probabilities within the data loader.

Training. To achieve our goal of maximizing the satisfaction level, we define the loss function as *1 - satisfaction_level*. In the training stage, we train the model for 500 epochs using the Adam optimizer. Every 20 epochs, we print out the satisfaction level and classification accuracy for both train and test data to track the learning outcomes.

3.3 Classical Classifiers

As a comparison to the LTNtorch implementation, classifiers are also applied on the same dataset, including SVM with different kernels (linear, polynomial, RBF), Logistic Regression, KNN, and Decision Tree with multiple settings (full-length tree, minimum number of leaves are 5). Specifically, we use KNN model with the best number of neighbors K found by computing the error rate under different K's (shown in Figure 2). Note that after K = 40, error rate doesn't go down, so we only show K up to 40 in the graph.

4 EXPERIMENTAL RESULTS AND ANALYSIS

The train and test accuracy for different classifiers are shown in the table. Since classical classifiers don't consider satisfaction level, we only compare accuracy scores.

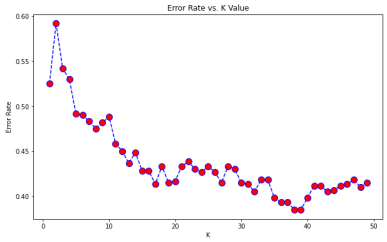

Figure 2: Error rate of KNN model under different number of nearest neighbors. The best (lowest error rate) case is when K = 38.

Model	Train accuracy	Test accuracy
LTNtorch	0.829	0.818
SVM (linear)	0.814	0.815
SVM (polynomial)	0.939	0.725
SVM (RBF)	0.929	0.783
KNN	0.633	0.615
LogReg	0.821	0.813
Full length tree	1.0	0.745
Min leaf 5 tree	0.903	0.74

Table 1: Train and test accuracy scores for all models under investigation.

LTNtorch has the highest test accuracy with 0.818 among all models. Although SVM (linear) and Logistic Regression models have very close test accuracy to LTNtorch, their train accuracy scores are lower then LTN. Thus, we still consider LTNtorch outperforms the other 2 models.

Model	Train accuracy	Test accuracy
LTNtorch	0.829	0.818
SVM (linear)	0.814	0.815
LogReg	0.821	0.813

Table 2: Comparison between LTN, SVM with linear kernel, and Logistic Regression.

Some models have the potential of overfitting, especially the full-length Decision Tree with the train accuracy 1.0. Thus, we consider these models have a lower performance.

Model	Train accuracy	Test accuracy
SVM (polynomial)	0.939	0.725
SVM (RBF)	0.929	0.783
Full length tree	1.0	0.745
Min leaf 5 tree	0.903	0.74

Table 3: Models that have the potential of overfitting.

Note that the KNN model perform poorly in both train (0.633) and test accuracy (0.615), so we don't consider it in this case.

In addition, although LTNtorch model already outperforms other models in this investigation, there are still some limitations of our model implementation. In our LTN setting, we simplify the input by treating the 20 features as a whole and only distinguishing between samples with different class labels without considering the relations/logic between the 20 features internally. To get more meaningful results, a break-down of the features and change the axioms to be feature-based should be explored.

5 CONCLUSION

In this paper, we investigate the Logical Tensor Networks on multi-class single label classification for the mobile phone price dataset, and compare the train and test accuracy with classical classifiers. LTN outperforms other classifiers and has no overfitting problems, which supports LTN's good ability in classification tasks.

6 FUTURE WORK

As stated previously, more exploration on the relations between the 20 phone features should be done to give more meaningful axioms and results.

Moreover, we hope to answer the question that if LTN consistently performs well in different dataset and tasks and what the reasons are behind this. Is this due to the certain properties of the dataset or specific tasks?

Also, we want to extend LTN to language processing fields such as text completion since there has little to none research been done in this application field.

REFERENCES

[1] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. 2021. Logic Tensor Networks. https://www.sciencedirect.com/science/article/pii/S0004370221002009
[2] Samy Badreddine and Michael Spranger. 2019. Injecting prior knowledge for transfer learning into reinforcement learning algorithms using logic tensor networks. https://arxiv.org/abs/1906.06576
[3] Federico Bianchi and Pascal Hitzler. [n.d.]. On the capabilities of logic tensor networks for deductive reasoning. https://people.cs.ksu.edu/~hitzler/pub2/aaaiss-19.pdf
[4] Bmxitalia. [n.d.]. Bmxitalia/ltntorch: Pytorch implementation of logic tensor networks, a neural-symbolic framework. https://github.com/bmxitalia/LTNtorch
[5] Tommaso Carraro. 2022. *LTNtorch: PyTorch implementation of Logic Tensor Networks.* https://doi.org/10.5281/zenodo.6394282
[6] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. 2016. Building machines that learn and think like people: Behavioral and brain sciences. https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/building-machines-that-learn-and-think-like-people/A9535B1D745A0377E16C590E14B94993
[7] Luciano Serafini, Ivan Donadello, and Artur d'Avila Garcez. 2017. Learning and reasoning in logic tensor networks: PROCEEDINGS OF THE SYMPOSIUM on applied computing. https://dl.acm.org/doi/pdf/10.1145/3019612.3019642
[8] Luciano Serafini and Artur S. D'Avila Garcez. 2016. Learning and reasoning with logic tensor networks: Proceedings of the XV international conference of the italian association for artificial intelligence on advances in artificial intelligence - volume 10037. https://dl.acm.org/doi/10.1007/978-3-319-49130-1_25
[9] Dongran Yu, Bo Yang, Dayou Liu, and Hui Wang. 2021. A survey on neural-symbolic systems. https://arxiv.org/abs/2111.08164
[10] Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. 2021. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. https://www.sciencedirect.com/science/article/pii/S2666651021000061