PiShield: A PyTorch Package for Learning with Requirements

Mihaela Cătălina Stoian\textsuperscript{1}, Alex Tatomir\textsuperscript{2}, Thomas Lukasiewicz\textsuperscript{2,1} and Eleonora Giunchiglia\textsuperscript{2}

\textsuperscript{1}Department of Computer Science, University of Oxford, UK
\textsuperscript{2}Institute of Logic and Computation, Vienna University of Technology, Austria

Abstract

Deep learning models have shown their strengths in various application domains, however, they often struggle to meet safety requirements for their outputs. In this paper, we introduce PiShield, the first package ever allowing for the integration of the requirements into the neural networks' topology. PiShield guarantees compliance with these requirements, regardless of input. Additionally, it allows for integrating requirements both at inference and/or training time, depending on the practitioners' needs. Given the widespread application of deep learning, there is a growing need for frameworks allowing for the integration of the requirements across various domains. Here, we explore three application scenarios: functional genomics, autonomous driving, and tabular data generation.

1 Introduction

Deep neural networks (DNNs) have shown their strengths in various application domains. However, they often fail to comply with given requirements defining the safe output space of the model. To obviate this problem, neuro-symbolic AI methods were introduced, which can be broadly classified into two categories. The first comprises methods able to integrate the requirements in the loss function and penalize the models when they violated the requirements [Diligenti \textit{et al.}, 2012; Diligenti \textit{et al.}, 2017; Donadello \textit{et al.}, 2017; Xu \textit{et al.}, 2018; Fischer \textit{et al.}, 2019; Nandwani \textit{et al.}, 2019; Badreddine \textit{et al.}, 2022; Li \textit{et al.}, 2023; Ahmed \textit{et al.}, 2022c; Stoian \textit{et al.}, 2023]). And while these approaches help reduce the requirements' violation incidence, they cannot guarantee their satisfaction. Emerging later, the second category consists of methods able to incorporate a given set II of requirements (also called constraints) directly in the topology of the network [Giunchiglia and Lukasiewicz, 2021; Hoermle \textit{et al.}, 2022; Ahmed \textit{et al.}, 2022b] and, thus, to guarantee their satisfaction. More recently, an alternative method [van Krieken \textit{et al.}, 2023], also capable of guaranteeing compliance with the constraints, proposed using neural networks for performing approximate inference in polynomial time to address the scalability problem of probabilistic neuro-symbolic learning frameworks such as DeepProbLog [Manhaeve \textit{et al.}, 2018]. For an in-depth survey of the methods combining deep learning with logical constraints, we refer to [Giunchiglia \textit{et al.}, 2022], while for a broad survey on neuro-symbolic AI, we refer to [d’Avila Garcez \textit{et al.}, 2019].

In this paper, we propose PiShield\textsuperscript{1,2}, a PyTorch-based package allowing for seamlessly integrating domain requirements into neural networks by means of new PyTorch layers that can be built on top of any neural network. These layers, which we call Shield Layers, adhere to the principles outlined in our most recent works [Stoian \textit{et al.}, 2024; Giunchiglia \textit{et al.}, 2024], advocating for a more requirements-driven machine learning as in [Giunchiglia \textit{et al.}, 2023a], and guarantee the satisfaction of the requirements regardless of the input. The Shield Layers can be applied during inference and/or training, depending on the practitioners’ needs. For users that have restricted access to a model but require that the outputs of their models are compliant with a set of rules, PiShield offers an easy-to-use interface to meet this need. Alternatively, PiShield can guide model training, suiting also practitioners accustomed to modifying their models. As expected, its ability to ensure compliance makes it ideal for safety-critical scenarios. Furthermore, studies [Giunchiglia and Lukasiewicz, 2020; Giunchiglia \textit{et al.}, 2024; Giunchiglia and Lukasiewicz, 2021; Stoian \textit{et al.}, 2024] have repeatedly shown its efficacy in aiding model learning. We illustrate both of these aspects on three different application domains: functional genomics, autonomous driving, and tabular data generation domains.

Related Work. A closely related work is Pylon [Ahmed \textit{et al.}, 2022a], a framework built on PyTorch which allows users to integrate constraints into a loss function. Similarly, the LTN package [Badreddine \textit{et al.}, 2022] provides a TensorFlow implementation of the Logic Tensor Networks (LTNs), while LTNTorch [Carraro, 2022] provides its PyTorch implementation. However, unlike our framework, all the methods described above cannot guarantee the satisfaction of the constraints.

2 PiShield Overview

PiShield is built on top of PyTorch, allowing for a seamless integration of Shield Layers into neural networks. Figure 1

\textsuperscript{1}Code: https://github.com/mihaela-stoian/PiShield
\textsuperscript{2}Website: https://sites.google.com/view/pishield

8805
Figure 1: PiShield overview.

gives an overview of how to apply PiShield to deep neural networks. As shown in the Figure, to constrain a DNN during training, the architecture of the DNN needs to be changed by appending a Shield Layer immediately after the output layer. To use PiShield only at inference time, it is enough to simply build a Shield Layer outside the neural network and apply it on the outputs to make them compliant with the requirements. Regardless of its use, each Shield Layer requires only two elements to be instantiated: (i) the dimension of the input to the layer (hence in most cases the dimension of the output of the network), and (ii) the path to the file which contains the requirements.

Requirements. The requirements can be expressed either as a propositional logic formula in conjunctive normal form (CNF) or as linear inequalities. In the file, each line should contain a single requirement, which will thus be either a clause in the first case or a single linear inequality in the latter.

Example 1. Suppose we have a simple multi-label classification problem where we are given as input images taken from an autonomous vehicle, and we have to identify whether a traffic light appears in the image, and whether it is green, yellow, or red. For each image, a standard neural network for this task will output a 4-dimensional vector where each element corresponds to one of the concepts of interest. Suppose, for example, the outputs of the neural networks are ordered in the following way: TrafficLight, Red, Yellow, Green. Then the file, containing the knowledge that a traffic light is always associated with one of the colors and that the colors associated with a traffic light are mutually exclusive, will have the format below:

\[
\text{not } y_0 \text{ or } y_1 \text{ or } y_2 \text{ or } y_3 \\
\text{not } y_0 \text{ or not } y_1 \text{ or not } y_2 \\
\text{not } y_0 \text{ or not } y_1 \text{ or not } y_3 \\
\text{not } y_0 \text{ or not } y_2 \text{ or not } y_3
\]

Example 2. Suppose we have a tabular data generation problem, where we have to generate a synthetic dataset for a clinical trial. Further, suppose that we have the following knowledge about the problem available: (i) the maximum hemoglobin recorded per patient should be always higher or equal than the minimum, and (ii) the maximum temperature recorded should be at least as high as the minimum. Then, assuming that the outputs of the neural network are ordered as MaxHemoglobin, MinHemoglobin, MaxTemp, MinTemp, the input file should have the following format:

\[
y_0 - y_1 \geq 0 \\
y_2 - y_3 \geq 0
\]

Usage. As it can be seen from both Listings 1 and 2, to build our layer, we only need (lines 5 and 6, respectively) to call the function build_shield_layer, which takes as input two parameters: (i) the dimension of the input to the layer, and (ii) the path to the requirements file. Once instantiated, the Shield Layer receives as input a tensor of predictions \(\hat{p}\) (possibly violating the requirements) and returns a tensor \(p\) (of the same dimension as \(p\)), which is now guaranteed to be compliant with the requirements. PiShield benefits from an easy-to-use interface, and so, correcting \(p\) with a Shield Layer is a one-step operation consisting of a forward call of the Shield Layer on \(p\). Listing 1, in particular, shows how to do all these steps at inference time. On the other hand, Listing 2 shows how to integrate the layer at training time for which the layer’s forward call needs to be done before the backpropagation step. As shown in the Listing, the easiest way to match this condition is to instantiate the Shield Layer in the constructor of the DNN and then call the layer inside the usual forward method (as shown in line 11).

3 Example Scenarios

The need for a package like PiShield naturally raises in many application domains. Here, we discuss three example scenarios to which we applied the layers implemented in PiShield.

Functional Genomics. In functional genomics, the task is to predict a set of (hierarchical) functions that genes may possess [Vens et al., 2008; Wehrmann et al., 2018; Clare, 2003]. As such, it belongs to the broad category of Hierarchical Multi-label Classification (HMC) problems, which are multi-label classification problems whose labels are organized in a hierarchical structure. The hierarchy can be captured by propositional logic rules, where the atoms correspond to the labels. In addition to functional genomics,
HMC problems find applications in many real-world domains, such as document (see e.g., [Klimt and Yang, 2004; Lewis et al., 2004]) and image classification (see e.g., [Deng et al., 2009]), or medical diagnosis (see e.g., [Dimitrovski et al., 2008]). In our demo video¹, we show on a simple task that applying PiShield to a model for this task guarantees that the outputs preserve the hierarchical structure.

Road Events Detection. Generalizing the above problem, we have the task of assigning sets of labels to instances of various types, including images, documents etc. Given available domain knowledge, constraints are placed on the possible outputs, eliminating impossible (i.e., contradicting background knowledge) outcomes. In our recent work [Giunchiglia et al., 2024], we considered the task of multi-label classification for an autonomous driving scenario using the ROAD-R dataset [Giunchiglia et al., 2023b], the first real-world dataset for autonomous driving with manually annotated logical constraints. ROAD-R is build upon the road event detection dataset ROAD [Singh et al., 2023] by adding 243 propositional logic requirements written in CNF. We showed that applying Shield Layers on the outputs is not only able to guarantee that these requirements are satisfied, but also results in increased performance, as seen in Table 2 of [Stoian et al., 2024]. Figure 2 shows examples of how using PiShield can impact the predictions’ quality, which is intrinsically coupled with the safety guarantees.

Tabular Data Generation. Another application where we utilized PiShield is the task of synthesizing tabular data such that the outputs of a model for this task satisfies background knowledge rules provided as linear inequalities, which capture relations between the features of the tabular data. Applying Shield Layers to standard deep generative models (DGMs) resulted in more realistic outputs, which are compliant with the background knowledge. For example, consider the constraint \( \text{MaxHaemoglobin} \geq \text{MinHaemoglobin} \), which is a real constraint that we encountered in [Stoian et al., 2024] in one of the datasets. In Figure 3, following the qualitative illustrations in Figures 3-6 of our previous work [Stoian et al., 2024], we show how the constrained DGMs match the real data more closely in the distribution of their outputs. Additionally, through an extensive empirical analysis, we showed (in Table 2 of [Stoian et al., 2024]) that integrating Shield Layers into DGMs yields an increased performance across two standard metrics used in tabular data generation.

Performance. In Table 1, we compare baseline models with their constrained versions, which use PiShield during training. We report results for three real-world applications: (i) functional genomics, based on Table 3 of [Giunchiglia and Lukasiewicz, 2020], reporting the area under the average precision and recall curve (AU(PRC)) over 8 datasets, (ii) autonomous driving, based on Table 2 of [Giunchiglia et al., 2024], reporting the frame-wise mean average precision (f-mAP) on the ROAD-R dataset, annotated with 243 propositional requirements; (iii) tabular data generation, based on Table 2 of [Stoian et al., 2024], reporting the F1-score for the utility performance (Utility-F1) averaged over 5 deep generative model types and 5 different datasets, annotated with up to 31 linear inequality constraints. As we can see, using PiShield during training provides major performance improvements over the unconstrained baselines.

4 Conclusions

In this paper, we introduced PiShield, the first package that allows for injecting requirements into neural networks’ topology by building Shield Layers, which correct the outputs so that they are guaranteed to satisfy the rules. PiShield can be applied (i) during training by building Shield Layers into the neural networks’ architecture, or (ii) at inference time by correcting the neural networks’ outputs as a post-processing step. We envision our package will be of use to practitioners working on real-world applications where domain knowledge can be expressed as propositional or linear constraints.

---

¹Video: http://tinyurl.com/pdv9eafa

Table 1: Aggregated performance. The best results are in bold.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Base.</th>
<th>PiShield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional genomics (AU(PRC))</td>
<td>0.225</td>
<td>0.241</td>
</tr>
<tr>
<td>Autonomous driving (f-mAP)</td>
<td>0.288</td>
<td>0.303</td>
</tr>
<tr>
<td>Tabular data generation (Utility-F1)</td>
<td>0.430</td>
<td>0.458</td>
</tr>
</tbody>
</table>
Ethical Statement

There are no ethical issues directly associating with the framework’s specification and available methods. Any ethical issues would only arise from the context in which the framework is planned to be used and on the constraints provided by the users. For example, when generating tabular data, the users could create a set of constraints that would facilitate (partially) recovering specific records in the training data, thus posing problems in cases where the data generated in this way can be made publicly available, but the training data cannot.

Acknowledgments

Mihaela Cătălina Stoian is supported by the EPSRC under the grant EP/T517811/1. This work was also supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1, by the AXA Research Fund, by the EPSRC grant EP/R013667/1, and by the EU TAILOR grant. We would like to thank Salija Dymishi, Thibault Simonetto, Andrew Ryzhikov, Salah Ghamizi, and Maxime Cordy for the useful discussions. We also acknowledge the use of the EPSRC-funded Tier 2 facility JADE (EP/P020275/1), GPU computing support by Scan Computers International Ltd.

References


[Hoermle et al., 2022] Nicholas Hoermle, Rafael-Michael Karampatsis, Vaishak Belle, and Kobi Gal. MultiplexNet:


