NSL: HYBRID INTERPRETABLE LEARNING FROM NOISY RAW DATA

A PREPRINT

Daniel Cunnington[1,2], Alessandra Russo[2], Mark Law[2], Jorge Lobo[2] and Lance Kaplan[3]

[1]IBM Research Europe, Winchester, UK., Contact: dancunnington@uk.ibm.com
[2]Imperial College London, London, UK
[3]Army Research Laboratory, Adelphi, MD, USA

December 10, 2020

ABSTRACT

Inductive Logic Programming (ILP) systems learn generalised, interpretable rules in a data-efficient manner utilising existing background knowledge. However, current ILP systems require training examples to be specified in a structured logical format. Neural networks learn from unstructured data, although their learned models may be difficult to interpret and are vulnerable to data perturbations at run-time. This paper introduces a hybrid neural-symbolic learning framework, called *NSL*, that learns interpretable rules from labelled unstructured data. NSL combines pre-trained neural networks for feature extraction with FastLAS, a state-of-the-art ILP system for rule learning under the answer set semantics. Features extracted by the neural components define the structured context of labelled examples and the confidence of the neural predictions determines the level of noise of the examples. Using the scoring function of FastLAS, NSL searches for short, interpretable rules that generalise over such noisy examples. We evaluate our framework on propositional and first-order classification tasks using the MNIST dataset as raw data. Specifically, we demonstrate that NSL is able to learn robust rules from perturbed MNIST data and achieve comparable or superior accuracy when compared to neural network and random forest baselines whilst being more general and interpretable.

1 Introduction

Inductive Logic Programming (ILP) systems learn (a set of) logical rules, that together with some background knowledge, explain a set of examples. ILP systems are often praised for their data efficiency [1, 2] and the interpretable nature of their learned rules [3]. Recent state-of-the-art ILP systems have also shown to be noise-tolerant and capable of learning complex knowledge from mislabelled data in an effective manner [4, 5]. A common characteristic of all state-of-the-art ILP systems is the fact that examples are required to be specified in a structured, logical format. This may hinder their applicability to real-world domains, leaving, as one of their main outstanding challenges, the ability to learn from unstructured data where contextual information may be noisy or perturbed.

Differentiable learning systems, such as deep neural networks [6], have demonstrated powerful function approximation on a wide variety of tasks, solving classification problems directly from unstructured data. However, large amounts of training examples are required, learned models are difficult to interpret [7] and may be vulnerable to distributional shift, where simple data perturbations are observed at run-time [8]. In this case, neural network predictions may be incorrect and the focus of this paper is learning robust rules in the presence of incorrect neural network predictions, taking into account the confidence score of such predictions.

To achieve this, we introduce a hybrid Neural-Symbolic Learning framework, called *NSL*, that aims to combine the advantages of both ILP and differentiable learning systems. NSL aims to learn a set of interpretable rules, that together with a (possibly empty) background knowledge, explain a given set of labelled unstructured data. It does so, by using a pre-trained neural network for extracting features from the unstructured data, and automatically generating structured

context-dependent examples that are used by a state-of-the-art ILP system, FastLAS [5] to learn rules that explain the given labelled unstructured data. The neural network predictions form the *context* of FastLAS's labelled examples and the confidence of the neural network predictions are aggregated to define a notion of penalty for these examples. FastLAS uses this notion of penalty, together with the length of rules, to define a cost over possible solutions. Rules that do not cover a context-dependent example pay the penalty of the example. Optimal solutions are computed as the set of rules with minimal cost. This enables NSL to cater for neural predictions with low confidence due to perturbations in the unstructured data and still learn rules that generalise (i.e. shorter rules pay a lower cost) and maximise coverage of labelled unstructured data with the highest aggregated confidence of neural predictions.

The NSL framework is evaluated on two classification tasks, a propositional animal classification task and a first-order valid Sudoku board classification task, where numerical features are extracted from MNIST digit images [9]. The neural network component is pre-trained on *non-perturbed* MNIST digits. In each task, the NSL framework is trained by perturbing the MNIST digits – rotating each digit image 90° clockwise and rules are learned that define animal class and the notion of valid Sudoku boards respectively. The learned rules are evaluated on structured ground truth test data and unstructured perturbed test data to, respectively (i) validate the accuracy of the learned rules with respect to a clean dataset (even though the rules were learned in the presence of perturbation at training time), and (ii) evaluate the robustness of the learned rules when applied to unseen perturbed unstructured data. In the first case and in both learning tasks, NSL is able to learn robust rules in the presence of training data perturbations, achieving comparable or superior accuracy than neural network and random forest baselines. Also, NSL learns rules that are more general and more interpretable. In the second case, when perturbations occur at run-time, NSL is able to maintain robust classification performance up to $\sim$40% perturbations, outperforming the neural network and random forest baselines. Finally, in the Sudoku classification task, we also demonstrate NSL's data efficiency compared to the neural network baseline, as the neural network requires 12.5X the number of examples required by NSL to achieve comparable performance.

The paper is structured as follows. In the next section we review relevant background material. We then introduce our NSL framework for learning rules from unstructured data, and discuss its algorithms. We then follow with results of our extensive evaluation. Finally we conclude with a discussion of related work.

2 Background

This section briefly introduces notations and terminologies used throughout the paper. An ILP learning task aims to find a set of rules, called a hypothesis, that explains a set of labelled examples [10]. Different approaches have been proposed in the literature [11] and in this paper we focus on the Learning from Answer Sets (LAS) approach [12], as the recently proposed ILASP systems have been shown to be robust to noisy examples [13] and scalable to large hypothesis spaces [5]. An Answer Set (ASP) program formalises a given problem in a logical form so that solutions to the program, called *answer sets*, provide solutions to the original problem. Formally, an ASP program is a set of rules of the form $\texttt{h} :- \texttt{b}_1 \ldots \texttt{b}_\texttt{n}, \texttt{not c}_1, \ldots \texttt{not c}_\texttt{m}$ where $\texttt{h}$, $\texttt{b}_\texttt{i}$ and $\texttt{c}_\texttt{j}$ are atoms; $\texttt{h}$ is the *head* of the rule, and $\texttt{b}_1 \ldots \texttt{not c}_\texttt{m}$ is the *body* of the rule, formed by a conjunction of positive literals ($\texttt{b}_1 \ldots \texttt{b}_\texttt{n}$) and negative literals ($\texttt{not c}_1 \ldots \texttt{not c}_\texttt{m}$) where $\texttt{not}$ is negation as failure. Given an ASP program P, the Herbrand Base of P (HB(P)) is the set of ground atoms constructed using the predicates and constants that appear in P. An interpretation I is a subset of HB(P). Given an interpretation I, the reduct of the grounding of an ASP program P, denoted as P^I (see [14] for definition and algorithm) is a grounded program with no negation as failure. I is an answer set of P if and only if it is the minimal model of the reduct program P^I. We denote the set of answer sets of a program P with $AS(P)$.

In the LAS framework, the objective is to learn ASP programs from examples that are *partial interpretations*. A *partial interpretation* e_{pi} is a pair of sets of ground atoms $\langle e^{inc}, e^{exc} \rangle$, called *inclusion* and *exclusion* sets respectively [5, 13]. An interpretation I *extends* e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. In ILASP [13], an example can have an associated context and penalty value. In this case, the example is called a weighted context-dependant partial interpretation (WCDPI). This is defined as a tuple $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$ where e_{id} is a unique identifier for e, e_{pen} is either a positive integer or ∞, called a *penalty*, e_{pi} is a partial interpretation and e_{ctx} is an ASP program relative to the example, called *context*. A WCDPI e is *accepted* by a program P if and only if there is an answer set of $P \cup e_{ctx}$ that extends e_{pi}. A task that learns ASP programs from WCDPIs is called a context-dependent LAS task and denoted as $ILP^{context}_{LAS}$. Such a task is defined as a tuple $\langle B, S_M, E \rangle$, where B is background knowledge, expressed as an ASP program, S_M is the hypothesis space, defined by a language bias M[1], and E is a set of WCDPIs. The hypothesis space defines the set of rules that can be used to construct a solution to the task. A hypothesis $H \subset S_M$ is a solution to a given $ILP^{context}_{LAS}$ task if and only if for every $e \in E$, H covers e, that is $B \cup e_{ctx} \cup H$ accepts e.

FastLAS is a system for solving a specific class of $ILP^{context}_{LAS}$ learning tasks, where for all $e \in E$, $|AS(B \cup e_{ctx})| = 1$ and no predicate in the head of a rule in S_M may occur in the body of any rule in S_M or in B [5]. Such tasks are referred

[1] For a detailed definition of a language bias see [4].

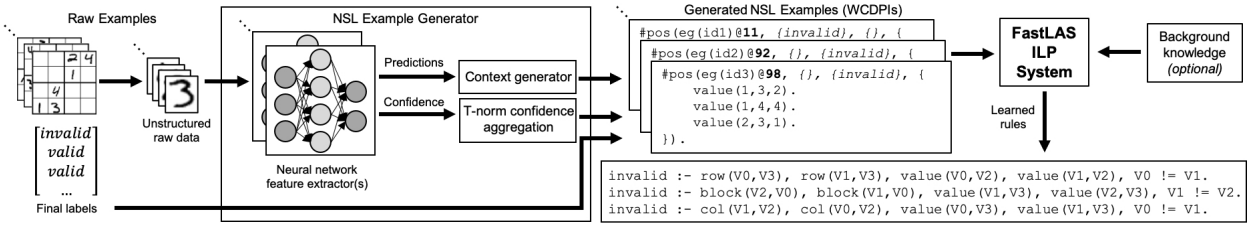


Figure 1: NSL framework with an example Sudoku board classification task

to in the literature as Observational Predicate Learning (OPL) tasks. In this paper we will denote them as ILP^{OPL}_{LAS}, and the set of possible solutions of a given ILP^{OPL}_{LAS} task LT, as $ILP^{OPL}_{LAS}(LT)$. Unless otherwise specified, we will assume that our ILP^{OPL}_{LAS} tasks have WCDPIs as examples.

The FastLAS system uses a *scoring function* to compute optimal solutions of a given learning task $LT = \langle B, S_M, E\rangle$. This is a function $\mathcal{S} : Programs \times T_{OPL} \to \mathbb{R}_{\geq 0}$ where $Programs$ is the set of all ASP programs and T_{OPL} is the set of all OPL tasks defined above (see [5] for a formal definition). A hypothesis $H \in Programs$ is said to be an *optimal solution* of a task $LT \in T_{OPL}$ with respect to a scoring function $\mathcal{S}$ iff $H \in ILP^{OPL}_{LAS}(LT)$ and there is no $H' \in ILP^{OPL}_{LAS}(LT)$ such that $\mathcal{S}(H', LT) < \mathcal{S}(H, LT)$. Scoring functions can be domain-specific and can be given as input to FastLAS prior to solving a learning task. FastLAS is capable of supporting any scoring function that is *decompositional*, that is for each $P \in Programs$ and $LT \in T_{OPL}$:

$$\mathcal{S}(P, LT) = \sum_{r \in P} \mathcal{S}_{rule}(r, LT)$$

where $\mathcal{S}_{rule} : Rules \times T_{OPL} \to \mathbb{R}_{\geq 0}$ is a function that scores individual rules of a solution H. We refer to $\mathcal{S}_{rule}$ as the decomposition of $\mathcal{S}$.

3 NSL Framework

This section introduces our Neural-Symbolic Learning (NSL) framework for learning rules from unstructured data. The framework uses two main computational components: a neural network, for extracting features from unstructured data, and FastLAS for learning rules from WCDPI examples. The bridge between these two components is realised through the *NSL example generator*, which converts unstructured labelled data into structured examples by generating a WCDPI example for FastLAS, from each set of neural predictions and a given label. A high level overview of the NSL framework, applied to a Sudoku classification task, is shown in Figure 1. The aim of the Sudoku task is to learn the classification of Sudoku boards as *valid* or *invalid*. The input is a set of images of digits in a Sudoku board with a valid/invalid label. The NSL framework returns as output an optimal set of rules that define the notion of an invalid Sudoku board. These rules can then be used to classify unseen Sudoku boards (given as sequences of digit images) as valid or invalid (see evaluation section). In what follows we present in detail the *NSL example generator* and the scoring function used by FastLAS as *optimisation criteria*, based on aggregated confidence of the neural network feature predictions. We then formalise the NSL learning task.

3.1 NSL Example Generator

The neural component of our NSL framework is a set of feature extractors $\mathcal{N}$, each pre-trained to extract a specific feature from a given piece of unstructured raw data. Let $d \in D$ be a piece of unstructured raw data (e.g. an image) and t be a string representing a specific feature of the unstructured raw data (e.g. a *digit*). A sample input x to the neural component is a pair $\langle d, t\rangle$ for which a feature extractor n_t exists in $\mathcal{N}$. For each sample input $x = \langle d, t\rangle$, the feature extractor $n_t \in \mathcal{N}$ outputs a k-dimensional probability vector ρ_d over the k-class feature t of d, i.e. $\rho_d = n_t(d)$. Within the scope of this paper, we assume that a specific feature extractor n_t can only extract feature t from a piece of unstructured raw data d of a given sample $x = \langle d, t\rangle$. Let X be a set of such samples.

We assume that each $n_t \in \mathcal{N}$ is a *pre-trained* neural network with fixed weights performing a classification task on input d. Each feature extractor n_t generates a prediction $y^t_{pred} = argmax(\rho_d)$ with a level of confidence $y^t_{conf} = max(\rho_d)$. These represent, respectively, the prediction of feature t and associated confidence estimation output by the neural network n_t for a given input d.

For each piece of unstructured raw data $d_i \in D$, the confidence score $y^{t_i}_{conf}$ of the corresponding feature extractor in $\mathcal{N}$ is used by the NSL example generator to define the penalty of a WCDPI example constructed from D. This is

achieved by means of a *general aggregation* function that combines a set of $y_{conf}^{t_i}$ confidence estimations into a single confidence score value. So, given a set of unstructured raw data D and an associated set of samples $X_D = \{\langle d_i, t_i \rangle\}$, the *overall confidence estimation* for D is an aggregation of the confidence estimation of each sample. This aggregated estimation, denoted as y_{conf}^{D}, constitutes a single confidence estimation value for a set of features extracted from unstructured raw data D.

The features extracted from unstructured raw data form the atomic building blocks for the *context* of the generated WCDPI examples, e_{ctx}. In FastLAS, contexts can be any ASP program which imposes additional example-based constraints that are taken into account during the learning task. In this paper, we consider a context to be a conjunction of atoms. To define the overall confidence estimation of a context e_{ctx} for an example e, NSL uses an aggregation function $\mathcal{A}$ defined in terms of triangular norms, (or *t*-norms) [15]. Given that a context of a generated example is a conjunction of atoms, NSL uses the Gödel *t*-norm ($\mathcal{T}_g(x, y) = min(x, y)$) ([16] and [17]) to recursively define the aggregation function $\mathcal{A}$. This is formally defined below. The aggregated confidence estimation y_{conf}^{D} defines the penalty value, e_{pen}, of a WCDPI example e, whose context e_{ctx} has been constructed from the predictions y_{pred}^{t} of a set of neural network feature extractors.

Definition 1 (General aggregation function)**.** Let $\mathcal{T}_g : [0, 1]^2 \rightarrow [0, 1]$ be the Gödel *t*-norm function defined as $\mathcal{T}_g(x, y) = min(x, y)$. Let $\mathcal{A}_z : [0, 1]^z \rightarrow [0, 1]$, where $z \geq 2$, be an aggregation function, recursively defined as follows[2]:

$$\mathcal{A}_2(x_1, x_2) = \mathcal{T}_g(x_1, x_2)$$
$$\mathcal{A}_z(x_1, \ldots, x_z) = \mathcal{T}_g(A_j(x_1, \ldots, x_j), A_{z-j}(x_{j+1}, \ldots, x_z))$$

A *general aggregation function* $\mathcal{A} : [0, 1]^z \rightarrow \mathbb{R}$ is then given by $\mathcal{A} = \lfloor \lambda \mathcal{A}_z \rfloor$ for $z \geq 2$, where λ is a parameter used alongside the floor function to convert the aggregated confidence score $\mathcal{A}_z([0, 1]^z) \in [0, 1]$ to an integer value.

Definition 2 (Context Generator)**.** Let F be the set of all possible sets of tuples $\langle y_{pred}, t, \alpha \rangle$ indicating a prediction y_{pred} for feature t. α can be used to encode additional information related to a particular feature prediction, such as the coordinates of a digit in a Sudoku board classification task. A context generator is a *function* $\mathcal{G} : F \rightarrow C$, where C is the set of all possible contexts for a WCDPI. Given an $f_i \in F$, $\mathcal{G}(f_i) = \{t(\alpha, y_{pred}). \mid \langle y_{pred}, t, \alpha \rangle \in f_i\}$.[3]

Note that $\mathcal{G}(f_i)$ gives the context e_{ctx} of an example for the FastLAS component based on the neural feature predictions.

Example 1. Let us assume a learning task with a set of unstructured raw data $D = \{$  ,  $\}$ and a set of features $T = \{digit, object\}$. Let us also assume a set of feature predictions $f = \{\langle 2, digit, \emptyset \rangle, \langle cat, object, \emptyset \rangle\}$ and a label l that corresponds to the set of unstructured raw data D. The context generator would then construct from the predictions f the context $e_{ctx} = \{$`digit(2).  object(cat).`$\}$.

Definition 3 (NSL example)**.** Let (ID, X, l) be an input tuple where ID is a unique identifier, X is a set of samples of unstructured raw data with associated set f_X of predicted features and a general aggregated confidence score A_X. Finally, let l be a label from a given set $\mathcal{L}$ of possible labels. An NSL example e is a WCDPI given by the tuple $\langle ID, A_X, e_{pi}, \mathcal{G}(f_X) \rangle$, where the partial interpretation e_{pi} is given by $\langle \{l\}, \{l^{'} \mid l^{'} \in \mathcal{L} \setminus \{l\}\} \rangle$.

Algorithm 1 generates the NSL examples from a given labelled sample of unstructured raw data.

Referring to Figure 1, the identifier ID for each NSL example is shown following the `#pos` statement and the NSL example penalty PEN calculated on line 13 in Algorithm 1 is shown in bold, immediately following the identifier. The NSL example penalty represents the aggregated confidence from neural network feature extractors. This encourages FastLAS to bias the learning towards hypotheses that cover NSL examples with high confidence neural network predictions, by ensuring that a high penalty is paid for leaving such examples uncovered by the learned hypothesis. The *inclusion* and *exclusion* sets (INC, EXC) respectively, are shown in italics[4] in Figure 1, followed by the *context* CTX given by the context generator $\mathcal{G}$.

3.2 NSL Optimisation Criteria

Our NSL framework uses the FastLAS system [5] for learning a hypothesis. FastLAS supports user-defined scoring functions that are capable of biasing the search for a hypothesis that is optimal with respect to a given domain-

[2]Due to the associativity property of T_g the recursive case is the same for all $1 \leq j \leq z$.

[3]Note that a more complex function $\mathcal{G}$ could be engineered, but this would require additional supervision.

[4]For binary classification tasks such as the Sudoku board classification task shown in Figure 1, we can simplify the inclusion and exclusion sets to only contain one label to represent the type of rules being learned. In this case, rules for *invalid* Sudoku boards are learned, so *valid* can be removed from the inclusion or exclusion set, depending on the example label.

Algorithm 1 NSL Example Generator

1: **procedure** GENERATE(ID, X, l)
2: $INC = \{l\}$;
3: $EXC = \{l^{'} \mid l^{'} \in \mathcal{L} \setminus \{l\}\}$;
4: $PRDCTS = []; CONFS = []$;
5: **for** $\langle d_i, t_i \rangle \in X$ **do**
6: $\rho_{d_i} = n_{t_i}(d_i)$;
7: $y^{t_i}_{pred} = argmax(\rho_{d_i})$;
8: $y^{t_i}_{conf} = max(\rho_{d_i})$;
9: $PRDCTS \mathrel{+}= (y^{t_i}_{pred}, t_i)$;
10: $CONFS \mathrel{+}= y^{t_i}_{conf}$;
11: **end for**
12: $CTX = \mathcal{G}(PRDCTS)$;
13: $PEN = \mathcal{A}(CONFS)$;
14: $WCDPI = (ID, PEN, (INC, EXC), CTX)$;
15: **return** $WCDPI$;
16: **end procedure**

specific optimisation criteria. Also, FastLAS supports learning from noisy examples, where the notion of noise can be domain-specific. NSL makes use of these two features of FastLAS to maximise example coverage whilst minimising the hypothesis length, such that learned rules are interpretable and generalise over unseen examples, whilst taking into account the confidence of the neural network predictions. Specifically, NSL considers the noise of an example to be the penalty value given by the general aggregation score of the example's context predicted by the neural network, and defines a scoring function that combines the standard ILP optimisation criteria, for minimising rule length to encourage generalisation, with an example penalty that caters for the confidence estimation of the neural network predictions.

Definition 4 (Penalty scoring function)**.** Let H be a hypothesis and E be a set of generated NSL examples. A penalty scoring function $\mathcal{S}_{penalty}(H, E) = \sum_{e \in UNCOV(H,E)} e_{pen}$ where e_{pen} is obtained from line 13 in algorithm 1, and $UNCOV(H, E)$ is the set of examples in E that are *not covered* by H. Note that H is said to cover an example e iff H accepts e [5].

The standard ILP scoring function [11] is given by the *length scoring function* $\mathcal{S}_{len}(H) = |H|$ which scores a hypothesis H simply by counting the number of literals in H.

As FastLAS supports user-defined scoring functions, in our NSL framework, we can define a scoring function that combines the standard length scoring function $\mathcal{S}_{len}$ with the penalty scoring function given in Definition 4, to jointly achieve both optimisation objectives. This is formally defined as follows:

Definition 5 (NSL scoring function)**.** Let H be a set of rules and E be a set of generated NSL examples. The NSL scoring function $\mathcal{S}_{NSL}(H, E) = \mathcal{S}_{penalty}(H, E) + \gamma \mathcal{S}_{len}(H)$ where $\gamma \in [0, \infty]$ can be used to weight example coverage or generalisation. In this paper, we assume $\gamma = 1$.

We can now define the notion of an NSL learning task. Informally, this is based on the notion of an Observational Predicate Learning task (ILP^{OPL}_{LAS}) under the Answer Set semantics [5], where examples are labelled unstructured data.

Definition 6 (NSL learning task)**.** An *NSL learning task* is a tuple $NSL_{task} = \langle B, S_M, \mathcal{S}_{NSL}, E^{RAW} \rangle$ where B is an ASP program called background knowledge, S_M is the set of possible rules, whose head predicates are the given labels and body predicates are predicates in B or in the given set of features T, $\mathcal{S}_{NSL}$ is the NSL scoring function and E^{RAW} is a set of labelled raw unstructured data.

The goal of the NSL learning task is to find an *optimal solution* to the task where the notion of *optimality* is given below.

Definition 7 (NSL Optimal Solution)**.** Let $LT = \langle B, S_M, \mathcal{S}_{NSL}, E^{RAW} \rangle$ be an NSL_{task} learning task. A set of rules $H \in S_M$ is an *optimal solution* of LT if and only if there is no $H' \in S_M$ such that $\mathcal{S}_{NSL}(H', E) < \mathcal{S}_{NSL}(H, E)$ where E is the set of NSL examples generated from the given E^{RAW}.

4 Evaluation

To evaluate our NSL framework, we firstly focus on the ability to learn generalised and interpretable rules in the presence of perturbed training data. Secondly, we evaluate predictive performance at run-time where test data is perturbed proportionally to the amount of perturbations present at training time.

Our evaluation is constructed over two neural-symbolic learning tasks:

1. **Zoo animal multi-class classification** where the goal is to classify an animal as either a *mammal*, *bird*, *fish*, *reptile*, *bug*, *amphibian*, or *invertebrate* from a variety of features such as the number of legs. For this task, we use the 'Zoo' data set[5] which contains 101 examples. We select 10 numeric features and substitute digit feature values for an image of a corresponding digit from the MNIST test set. This task can be solved using propositional rules.
2. **Sudoku board binary classification** as illustrated in Figure 1 where the goal is to learn rules that classify a 4x4 Sudoku board as *valid* or *invalid*. Hanssen's Sudoku puzzle generator[6] is used to generate 200 valid boards. We generate a further 200 invalid boards manually. In this task, digits on the Sudoku board are also replaced with images of corresponding MNIST test set digits. This task requires first-order rules (i.e. rules with variables) to correctly differentiate between valid and invalid Sudoku boards.

We perform 5-fold cross-validation in both learning tasks with an 80%/20% train/test split. Full details regarding the data sets are listed in Section 4.3.

For the zoo animal classification task we compare our NSL framework to a shallow feed forward neural network and random forest baselines[7], as neural networks and tree-based methods are known to perform well on this task [18, 19]. Also, their learned models can be evaluated for interpretability by either using a surrogate model to approximate the predictions of a neural network, or inspecting learned trees directly in the case of a random forest.

For the Sudoku board classification task we compare NSL to a CNN-LSTM deep neural network architecture due to the spatial dependency between the sequence of digits on a Sudoku board. This architecture has demonstrated strong performance on other sequence classification tasks [20, 21]. We also compare NSL to a random forest model due to its inherently explainable properties.

Our evaluation uses two neural network feature extractors pre-trained on the MNIST training set. Their weights remain fixed throughout our experiments. We use a LeNet 5 architecture [9] with the standard softmax classification layer and also a state-of-the-art *uncertainty-aware* network called *EDL-GEN* [22][8]. To ensure a fair comparison with our NSL framework, in both learning tasks, all neural network and random forest baselines are given softmax feature extractor predictions as input and don't learn over the raw unstructured data directly. When aggregating softmax or EDL-GEN confidence information within the construction of an NSL example, we use the Gödel t-norm aggregation function due to the conjunctive nature of the learning tasks presented in this paper. The classification label in both tasks depends on the conjunction of animal features, or the conjunction of digits on a Sudoku board.

In both learning tasks, we simulate distributional shift for the neural network feature extractors by perturbing all the unstructured raw data within an increasing percentage of training examples. Unstructured raw data is perturbed by rotating each MNIST image within an example 90° clockwise [8][9]. The final labels (i.e. the animal classification or the Sudoku board classification) remain unchanged.

4.1 Learning rules from perturbed training data

Figures 2a and 2c show the results for learning rules from perturbed training data on the zoo and Sudoku tasks respectively. In both tasks, the evaluation is performed on ground truth test data where the original feature values are used and are *not* substituted for their corresponding MNIST digits. This is to indicate the quality of the learned rules and to evaluate the ability to ignore data perturbations at training time. The mean classification accuracy across all 5 cross-validation splits is reported to highlight classification performance. The standard error of the mean across all cross-validation splits is shown with the displayed error bars to indicate the deviation between the sampled mean

[5] `https://archive.ics.uci.edu/ml/datasets/Zoo`

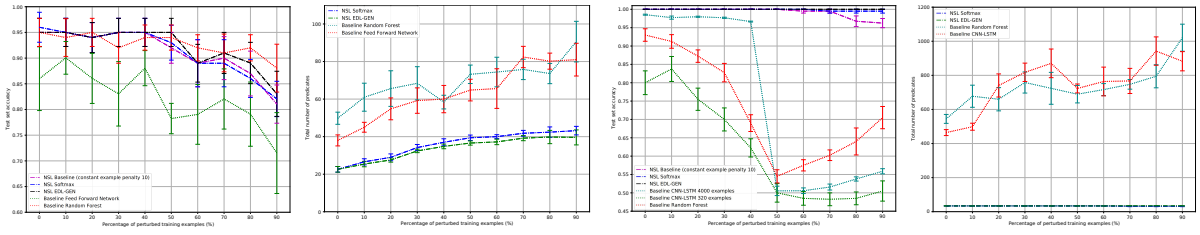
[6] `https://www.menneske.no/sudoku/2/`

[7] Details of selected hyper-parameters and model architectures are given in Section 4.4

[8] We use the softmax and EDL-GEN feature extractor implementation from `https://muratsensoy.github.io/uncertainty.html` and `https://muratsensoy.github.io/gen.html` respectively.

[9] Note. This is one example of a distributional shift.

and the true population mean. The *NSL Baseline* uses a constant example penalty of 10 within FastLAS instead of aggregating neural network feature extractor confidence information.



(a) Zoo - accuracy (b) Zoo - Interpretability (c) Sudoku - accuracy (d) Sudoku - interpretability

Figure 2: Accuracy of learned rules with structured ground truth test data and interpretability of learned rules for both the zoo animal classification and Sudoku board classification tasks

In terms of ground truth test set classification performance, NSL out-performs the baseline methods in both the zoo and Sudoku learning tasks, with the exception of the random forest in Figure 2a. Aggregating confidence information from neural network feature extractors leads to a benefit in both learning tasks after $\sim$40% perturbations, with the EDL-GEN feature extractor outperforming softmax.

The zoo task in Figure 2a is relatively simple for symbolic ILP learners and the neural network baseline even in a simple task is vulnerable to data perturbations, as NSL clearly demonstrates superior performance. NSL is marginally beaten by the random forest at 20% and beyond 50% perturbations, although NSL is within the standard error of the random forest. In order to obtain this increase in performance, the random forest learns a less interpretable model. To evaluate interpretability, we obtain the total number of predicates across all learned rules and plot the mean across all 5 cross-validation splits. The standard error of the mean across all splits is indicated with error bars. Rules with a lower number of predicates are assumed to be more interpretable [23]. Figures 2b and 2d show the total number of learned predicates increases as more perturbations are present in the training examples. In both the zoo and Sudoku learning tasks, NSL learns a more interpretable set of rules than the baseline methods for all percentages of training example perturbation and NSL with an EDL-GEN feature extractor outperforms NSL with a softmax feature extractor.

We note that explaining what the random forest has learned may not be trivial, depending on the number of trees in the forest. In our experiments we have 100 trees in the ensemble. To interpret the neural network models, a surrogate decision tree is trained to approximate the neural network predictions [24]. For the random forest models, rules learned from the first tree within the ensemble are selected.

In Figure 2c, which is a more complicated task requiring the use of memory, NSL outperforms all baselines for all percentages of training example perturbations. NSL is able to take advantage of background knowledge regarding the layout of a Sudoku board to increase its performance. At 50% perturbations the performance of all the neural network and random forest baselines reduce to $\sim$50% and start to increase again as more patterns begin to emerge in the training data. Given the CNN-LSTM is a deep neural network architecture, we conducted a further experiment with an increased number of training examples. We found that at 4000 training examples the CNN-LSTM was able to achieve similar performance to NSL with 320 training examples below 50% example perturbations. This is a 12.5X reduction, indicating the data efficiency of the NSL framework as a result of background knowledge integration within FastLAS.

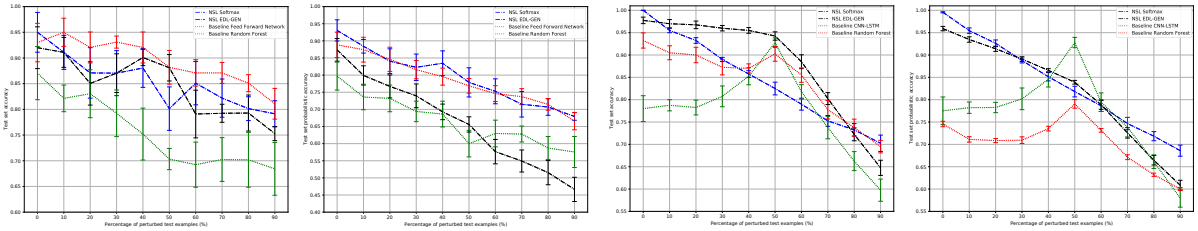
4.2 Run-time perturbations

This subsection presents results where perturbations are applied to unstructured data at run-time in the same manner and at the same proportion that was observed during training in Figure 3, for both the zoo and Sudoku learning tasks.

In order to account for unconfident neural network predictions at run-time, we convert the rules learned with NSL to a ProbLog program [25] and create annotated disjunctions [26] to represent the probability of a given MNIST image being classified as a certain digit. ProbLog then outputs a prediction as well as a probability of the prediction being true with respect to the neural network feature extractions and the learned rules. This is interpreted as a confidence value to align with the baseline neural network and random forest methods which already contain built-in confidence outputs at run-time.

To evaluate both the prediction and the confidence simultaneously, in addition to the standard accuracy metric, we use a modified accuracy metric, denoted *probabilistic accuracy*, or *prob. accuracy* for short, calculated as $\hat{a} = \frac{\sum_{i=1}^{n} p^i_{y\_true}}{n}$

where $p^i_{y\_true}$ represents the predicated confidence for the *ground-truth* class for the ith test example and n is the number of test examples. Instead of simply counting a correct (resp. incorrect) prediction as 1 (resp. 0) as with standard accuracy, this ensures that the learner will be penalised for predicting correctly with low confidence and also predicting incorrectly with high confidence. Error bars indicate the standard error when evaluating learned models from all 5 cross-validation splits. For these experiments, the *NSL Baseline* is removed given we are evaluating in the presence of perturbed test data and unconfident neural network predictions.



(a) Zoo - accuracy (b) Zoo - prob. accuracy (c) Sudoku - accuracy (d) Sudoku - prob. accuracy

Figure 3: Run-time perturbation results for both the Zoo and Sudoku learning tasks, evaluated with accuracy and prob. accuracy

For the zoo experiments in Figure 3a, both NSL approaches outperform a neural network baseline and when evaluating with probabilistic accuracy in Figure 3b, NSL softmax outperforms the random forest apart from 20%, 60%, 70% and 80% perturbations. In the more complex task of learning rules for invalid Sudoku boards, both NSL approaches outperform the baselines until 40% perturbations in both Figure 3c and 3d, with the NSL EDL-GEN outperforming all approaches until 80% perturbations in Figure 3c. Comparing Figures 3c and 3d with Figure 2c, it's likely that at 50% perturbations the neural network has over-fitted to the pattern of perturbations in the training data and hasn't learned the true rules, given its strong performance when test data is perturbed and low performance when the test data is not perturbed.

4.3 Data set Preparation

1. **Zoo animal classification** This data set is obtained from the UCI Machine Learning repository and contains 101 examples. In our experiments, we select the *hair*, *feathers*, *eggs*, *milk*, *aquatic*, *predator*, *fins*, *legs* and *tail* features to enable experiments to be completed in a timely manner. The goal is to classify an example as either a *mammal*, *bird*, *fish*, *reptile*, *bug*, *amphibian*, or *invertebrate*. We perform 5-fold cross validation to generate 5 training sets each containing 80 examples and 5 test sets each containing 21 examples respectively. Digit features are then substituted for a random image from the MNIST test set, corresponding to each digit. This is initially performed for the 5 zoo training sets and for the run-time perturbation experiments this is performed for the 5 zoo test sets also.
2. **Sudoku board classification** This data set is generated using 4x4 Sudoku puzzles obtained from Hanssen's Sudoku puzzle generator. We obtain 200 valid puzzles and generate a further 200 invalid puzzles manually to create a data set of 400 examples. The goal is to classify whether a Sudoku board is *valid* or *invalid* based on the digits present in the board. Each example contains a varying number of digits and no example details a completed board. Similarly to the zoo animal classification data set, we perform 5-fold cross validation to generate 5 training sets each containing 320 examples and 5 test sets each containing 80 examples respectively. When evaluating the CNN-LSTM architecture with a larger data set, we obtain 2500 valid Sudoku puzzles from Hanssen's Sudoku puzzle generator and generate a further 2500 invalid puzzles manually. 5-fold cross-validation is then performed to generate 5 training sets of 4000 examples and 5 test sets of 1000 examples respectively. We also perform a similar substitution of digits with random images from the MNIST test set corresponding to the same digit.

4.4 Baseline Architectures and Hyper-Parameter Tuning

1. **Zoo animal classification** The baseline feed forward network consists of 3 fully connected layers where the first two layers contain *ReLU* activation functions and the final layer contains a *softmax* activation function. The number of neurons in the first two layers as well as the batch size were tuned using a grid search with the following parameter values. Number of neurons in layer 1: $\{5, 12, 20\}$, number of neurons in layer 2: $\{8, 10, 15\}$ and batch size: $\{2, 5, 10\}$. Tuning was performed over the extended zoo dataset '*zoo3*'[10] with

[10]https://www.kaggle.com/agajorte/zoo-animals-extended-dataset

no MNIST digit substitution where the best performing combination of parameters were chosen according to test set accuracy. We performed an 80%/20% train/test split on the zoo3 data set using the scikit-learn library. All random seeds were set to 0 and the chosen parameters were 12 neurons in the first layer, 8 neurons in the second layer and a batch size of 10. The model was trained for 150 epochs with the sparse categorical cross-entropy loss using the Adam optimiser and was implemented in TensorFlow 2.3.0 with Keras 2.4.0.

The baseline random forest model was implemented with scikit-learn 0.23.2 and tuned on the same train/test split of the zoo3 dataset as the feed forward neural network. The number of estimators were tuned across: $\{10, 50, 100, 200\}$. The best performing parameter value of 100 estimators was chosen.

2. **Sudoku board classification** The baseline CNN-LSTM network consists of an embedding layer, followed by a 1D convolutional layer with 32 filters, a kernel size of 3, the ReLU activation function and 'same' padding. Then, a 1D max pooling layer with pool size 2 is used, followed by a dropout layer with dropout probability 0.2, an LSTM layer and a second dropout layer also with dropout probability 0.2. Finally, a dense fully connected layer with the sigmoid activation function is used to produce a binary classification of the input digit sequence. The input sequence length to the embedding layer is 16, representing each cell on the 4x4 Sudoku board.

   Tuning was performed using a separate dataset of 100 valid and 100 invalid Sudoku boards using a grid search to tune the embedding vector length, the number of neurons in the LSTM layer and the batch size with the following parameter values. Embedding vector length: $\{10, 20, 32\}$, number of neurons in the LSTM layer: $\{50, 100\}$ and batch size: $\{8, 16, 32\}$. The best performing combination of parameter values was chosen according to test set accuracy. We performed an 80%/20% train/test split using the scikit-learn library. All random seeds were set to 0 and the chosen parameters were an embedding vector length of 32, 100 neurons in the LSTM layer and a batch size of 32. The model was trained for 200 epochs with the binary cross-entropy loss using the Adam optimiser and was implemented in TensorFlow 2.3.0 with Keras 2.4.0.

   The baseline random forest model was implemented with scikit-learn 0.23.2 and tuned on the same train/test split of the separate Sudoku dataset as the CNN-LSTM neural network. The number of estimators were tuned across: $\{10, 50, 100, 200\}$. The best performing parameter value of 100 estimators was chosen.

4.5 System Details

All experiments in this paper were run on the same machine with the following specifications:

- **Hardware:** QEMU KVM virtual machine standard PC (i440FX + PIIX 1996) with 10 nodes of 8-core AMD EPYC Zen 2 CPUs (80 cores total), 16GB RAM.
- **Operating System:** Ubuntu 18.04.4 LTS.
- **Software:** FastLAS 1.0, Python 3.7.3, TensorFlow 2.3.0, Keras 2.4.0, scikit-learn 0.23.2, numpy 1.19.1, problog 2.1.0.42.

5 Related Work

Existing approached for integrating neural and symbolic techniques can be categorised into two main types. Firstly, there are a set of approaches that leverage neural networks for learning and symbolic components for reasoning. Secondly, there are approaches that leverage both neural networks and symbolic components together to perform a joint learning task.

DeepProbLog [27] is an example of an integrated neural-symbolic learning and reasoning framework that connects probabilistic logic to neural networks to enable probabilistic reasoning over neural network outputs. Neural network feature extractors within DeepProbLog can be trained w.r.t resulting policy labels, by constructing a Sentential Decision Diagram (SDD) based on hard-coded logical rules. DeepProbLog requires these logical rules to be manually specified and they are not learned from data.

Logic Tensor Networks (LTNs) [28] generalise the semantics of first-order logic to introduce real logic, replacing the standard Boolean values with real values in the interval $[0, 1]$. LTNs enable integrated learning through tensor networks and reasoning using real logic. This enables soft and hard logical constraints and relations to be specified in first-order logic as background knowledge to help guide neural networks during training. With LTNs, logical rules aren't learned symbolically, learning occurs in a neural network constrained by real logic.

In both of these approaches, logical rules are not directly learned from unstructured data, which is the key focus of our NSL framework. A differentiable ILP framework [29] called ∂ILP learns rules from unstructured data by re-implementing a top-down, generate and test ILP approach. ∂ILP learns, through stochastic gradient descent, which

generated clauses should be "turned on" such that together with the background knowledge, the turned on clauses entail the positive examples and do not entail the negative examples. Their evaluation includes a pre-trained neural network that classifies MNIST digits connected to an ILP system for rule learning. Their approach requires, a set of predefined rule templates to define the hypothesis space. This is memory intensive and the approach is limited to predicates up to arity 2. NSL does not have any such constraints as FastLAS is more scalable than the ILP system used in ∂ILP in terms of the number of examples and the size of the hypothesis space. For example, ∂ILP cannot be used on the Mutagenesis data set[11], which contains 230 examples, whereas the Sudoku task presented in this paper contains 320 training examples. Finally, the differentiable integration in ∂ILP is tightly coupled with the ILP system, whereas NSL is modular and preserves separation between the neural and symbolic components.

6 Conclusion

This paper has introduced a hybrid neural-symbolic learning framework called *NSL* that is capable of learning robust rules in the presence of perturbed training data and incorrect or unconfident neural network predictions. The framework aggregates confidence information from neural network feature extractors to create a penalty paid by the FastLAS ILP system for not covering an NSL example. Our results indicate that NSL is able to learn robust rules in the presence of perturbed training data, matching or outperforming the neural network and random forest baselines whilst learning a more general and interpretable model. When perturbations occur at run-time, NSL is able to maintain robust classification performance up to $\sim$40% perturbations, outperforming the neural network and random forest baselines. Finally, in the Sudoku task, FastLAS can utilise background knowledge to learn robust rules with fewer training examples compared to a neural network baseline.

7 Ethics Statement

Interpretable machine learning systems provide a first step towards addressing issues such as bias, discrimination and accountability. In practical terms, it often isn't possible to acquire training data that perfectly represents the intended distribution within a population, one can only approximate the distribution through a set of samples. Powerful machine learning algorithms, such as deep neural networks may unwillingly amplify and reinforce inherent bias present in the sampled underlying training data that can be difficult to detect until the model is deployed.

In this paper, we explore the combination of an interpretable ILP system with a deep neural network feature extractor such that learned rules from perturbed data can be inspected for correctness and potential bias or discrimination. An interesting example of the benefit of this approach can be observed in the Sudoku board classification task within our evaluation. Referring to Figures 3c and 3d when evaluating on perturbed test data, the CNN-LSTM neural network baseline appears to perform strongly, given 50% perturbed training and test examples. However, comparing the performance of the same model when evaluated on *non-perturbed* test data in Figure 2c, the performance is significantly reduced. This indicates the CNN-LSTM has amplified the perturbations present in the training data. This particular model may be difficult to interpret which highlights a potential issue should the training data contain biased or discriminatory examples. An interpretable approach, such as the NSL framework presented in this paper, could highlight such amplifications of biased training data before the model is deployed.

8 Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Andrew Cropper, Sebastijan Dumancic, and Stephen H Muggleton. Turning 30: New ideas in inductive logic programming. In *Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

[11] https://relational.fit.cvut.cz/dataset/Mutagenesis

[2] Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B Tenenbaum, and Stephen H Muggleton. Bias reformulation for one-shot function induction. *Frontiers in Artificial Intelligence and Applications*, pages 525–530, 2014.

[3] Stephen Muggleton, Schmid Ute, Tamaddoni-Nezhad Alireza, and Besold Tarek. Ultra-strong machine learning: comprehensibility of programs learned with ilp. *Machine learning*, pages 1119–1140, 2018. Special Issue of the Inductive Logic Programming (ILP) 2016.

[4] Mark Law. *Inductive learning of answer set programs*. PhD thesis, Imperial College London, 2018.

[5] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. In *AAAI*, 2020.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, 2018.

[8] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems*, pages 3179–3189, 2018.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[11] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[12] Mark Law, Alessandra Russo, and Krysia Broda. Logic-based learning of answer set programs. In *Reasoning Web. Explainable Artificial Intelligence - 15th International Summer School 2019, Bolzano, Italy, September 20-24, 2019, Tutorial Lectures*, pages 196–231, 2019.

[13] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*, 2018.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, volume 88, pages 1070–1080, 1988.

[15] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular norms*, volume 8. Springer Science & Business Media, 2013.

[16] George Metcalfe, Nicola Olivetti, and Dov M Gabbay. *Proof theory for fuzzy logics*, volume 36. Springer Science & Business Media, 2008.

[17] Qun Ni, Elisa Bertino, and Jorge Lobo. Risk-based access control systems built on fuzzy inferences. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, page 250–260, New York, NY, USA, 2010. Association for Computing Machinery.

[18] Ibrahim M Nasser and Samy S Abu-Naser. Artificial neural network for predicting animals category. *International Journal of Academic and Applied Research (IJAAR)*, 2019.

[19] Ratula Ray and Satya Dash. Comparative study of the ensemble learning methods for classification of animals in the zoo. In *Smart Computing and Informatics (SCI) Conference*, pages 251–260, 12 2018.

[20] A. Yenter and A. Verma. Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis. In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pages 540–546, 2017.

[21] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

[22] Murat Sensoy, Lance Kaplan, Federico Cerutti, and Maryam Saleki. Uncertainty-aware deep classifiers using generative models. In *AAAI*, 2020.

[23] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1675–1684, 2016.

[24] Christoph Molnar. *Interpretable Machine Learning*. Lulu.com, 2020.

[25] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 2462–2467, 01 2007.

[26] Dimitar Shterionov, Joris Renkens, Jonas Vlasselaer, Angelika Kimmig, Wannes Meert, and Gerda Janssens. The most probable explanation for probabilistic logic programs with annotated disjunctions. In *Inductive Logic Programming*, pages 139–153. Springer, 2015.

[27] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

[28] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.

[29] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.