FF-NSL: Feed-Forward Neural-Symbolic Learner*  

Daniel Cunnington · Mark Law · Alessandra Russo · Jorge Lobo  

Abstract Inductive Logic Programming (ILP) aims to learn generalised, interpretable hypotheses in a data-efficient manner. However, current ILP systems require training examples to be specified in a structured logical form. This paper introduces a neural-symbolic learning framework, called Feed-Forward Neural-Symbolic Learner (FF-NSL), that integrates state-of-the-art ILP systems based on the Answer Set semantics, with neural networks, in order to learn interpretable hypotheses from labelled unstructured data. FF-NSL uses a pre-trained neural network to extract symbolic facts from unstructured data and an ILP system to learn a hypothesis that performs a downstream classification task. In order to evaluate the applicability of our approach to real-world applications, the framework is evaluated on tasks where distributional shifts are introduced to unstructured input data, for which pre-trained neural networks are likely to predict incorrectly and with high confidence. Experimental results show that FF-NSL outperforms baseline approaches such as a random forest and deep neural networks by learning more accurate and interpretable hypotheses with fewer examples.

Keywords Neural-Symbolic Learning · Inductive Logic Programming · Distributional shift

* Pre-print, work in progress

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.

Daniel Cunnington  
IBM Research Europe - Hursley, Winchester, UK  
Imperial College London, London, UK  
E-mail: dancunnington@uk.ibm.com

Mark Law, Alessandra Russo, Jorge Lobo  
Imperial College London, London, UK
1 Introduction

Inductive Logic Programming (ILP) systems learn a set of logical rules, called a hypothesis, that together with some (optional) background knowledge, explains a set of labelled examples [24]. ILP systems are often praised for their data efficiency [4,20] and the interpretable nature of their learned hypotheses [25]. State-of-the-art ILP systems have also been shown to be capable of learning complex knowledge from noisy examples [15,16]. However, ILP systems require examples to be specified in a structured logical form, which limits their applicability to real-world tasks.

Differentiable learning systems, such as (deep) neural networks [12], have demonstrated powerful function approximation on a wide variety of real-world problems, learning directly from unstructured data. These approaches, however, are vulnerable to distributional shifts, where data observed at run-time belongs to a different distribution than that observed during training, leading to incorrect predictions with potentially high confidence [27,32,1]. Furthermore, such approaches tend to require large amounts of training data and learn models that are difficult to interpret [11].

This paper introduces a neural-symbolic learning framework, called Feed-Forward Neural-Symbolic Learner (FF-NSL), that aims to address the drawbacks of these two paradigms by integrating pre-trained neural networks, to extract symbolic facts from unstructured data, with state-of-the-art ILP systems, such as ILASP [15] and FastLAS [16], to learn generalised and interpretable hypotheses that can solve a downstream classification task. This seamless integration enables pre-trained neural networks to be used, exploiting the ILP systems’ ability to learn complex knowledge from noisy examples, such that incorrect neural network predictions, (potentially made with high confidence) can be tolerated. Therefore, FF-NSL is able to learn accurate and interpretable hypotheses from unstructured input data subject to distributional shifts, in comparison to the distribution of data originally used to pre-train the neural network.

To enable the integration of neural and symbolic components, FF-NSL automatically maps symbolic facts extracted by a neural network into training examples for the ILP system, whilst preserving the level of confidence of the neural network predictions. This is achieved by an example generator function that converts symbolic facts extracted from unstructured data, into relevant contexts for ILP examples. The level of noise of such examples is defined in terms of the confidence score of the neural network predictions. The modularity of the framework enables different neural network architectures, including state-of-the-art uncertainty-aware neural networks (e.g., [31]) to be used alongside state-of-the-art ILP systems, provided that the ILP system is robust to noisy examples.

To evaluate the accuracy and interpretability of our FF-NSL approach, as well as the robustness to distributional shifts in input unstructured data, FF-NSL is evaluated on two classification tasks:

1) Sudoku grid validity, where the objective is to learn a hypothesis that defines an invalid Sudoku grid. Digits in the Sudoku grid are represented with images of MNIST digits and distributional shift is achieved by rotating MNIST digit images
90° clockwise in an increasing percentage of Sudoku grid examples. The approach is demonstrated on 4x4 and 9x9 Sudoku grids.

2) **Follow Suit winner**, where the objective is to learn a hypothesis that defines the winner of a four-player, trick-based card game called **Follow Suit**. The winner of each trick is the player that plays the highest ranked card with the same suit as Player 1. We represent playing cards using images from a standard deck and distributional shift is achieved by replacing standard card images with images from alternative decks in an increasing percentage of Follow Suit example tricks.

Our experimental results demonstrate that FF-NSL learns an accurate, general and interpretable hypothesis given unstructured input data subject to distributional shifts. In particular, FF-NSL (1) outperforms baseline random forest and deep neural network architectures on both tasks, with regards to the accuracy and interpretability of the learned rules. (2) Learns with 100X fewer examples. (3) When applied at run-time to unseen unstructured data, FF-NSL outperforms baseline methods trained with the same number of examples, where distributional shift is applied up to ∼80% of input data. Finally, (4) calculating ILP example weight penalties from neural network confidence scores leads to improved performance in comparison to constant example penalties, and, (5) when trained using a state-of-the-art uncertainty-aware neural network ([31]), FF-NSL achieves superior accuracy and faster learning time.

The paper is structured as follows. Section 2 summarises the necessary background notation regarding the Learning from Answer Sets (LAS) framework [18] and the related ILP systems used in this paper. Section 3 introduces the FF-NSL framework and Section 4 presents our evaluation results. Related work is discussed in detail in Section 5 and we conclude in Section 6.

### 2 Background

An ILP learning system aims to find a set of logical rules, called a **hypothesis** that, together with some background knowledge, explains a set of labelled examples [24]. Different approaches have been proposed in the literature [28]. In this paper, we build upon the LAS framework [18], and its ILP systems ILASP [15] and Fast-LAS [16], which have shown to be robust to noisy data [17] and scalable to large hypothesis spaces [16].

Answer Set Programming (ASP) formalises a given problem in a logical program so that solutions to the program, called **answer sets**, provide solutions to the original problem. An ASP program is a set of rules of the form $h :- b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m$ where $h, b_i$ and $c_j$ are atoms; $h$ is the **head** of the rule, and $b_1, \ldots, \text{not } c_m$ is the **body** of the rule, formed by a conjunction of positive literals $(b_1, \ldots, b_n)$ and negative literals $(\text{not } c_1, \ldots, \text{not } c_m)$ where not is negation as failure [3]. Given an ASP program $P$, the Herbrand Base of $P$ ($HB(P)$) is the set of ground (variable free) atoms constructed using the predicates and constants that appear in $P$. An interpretation $I$ is a subset of $HB(P)$. A ground rule $R$ is satisfied by an interpretation $I$ if and only if the head of $R$ is satisfied by $I$ or the body of $R$ is not. Any $I \subseteq HB(P)$ is a model if it satisfies every ground instance of every rule

---

1. [https://en.wikipedia.org/wiki/Trick-taking_game](https://en.wikipedia.org/wiki/Trick-taking_game)
in $P$. The answer sets of a program $P$ are a special subset of the models of $P$ (for a formal definition, see [14]).

In the LAS framework, the objective is to learn ASP programs from examples, defined in terms of partial interpretations [17]. A partial interpretation $e_{\text{pi}}$ is a pair of sets of ground atoms $(e_{\text{inc}}^e, e_{\text{exc}}^e)$, called inclusion and exclusion sets respectively. An interpretation $I$ extends $e_{\text{pi}}$ if $e_{\text{inc}}^e \subseteq I$ and $e_{\text{exc}}^e \cap I = \emptyset$. An example $e$, referred to as a Weighted Context-Dependant Partial Interpretation (WCDPI), is a tuple $(e_{\text{id}}, e_{\text{pen}}, e_{\text{pi}}, e_{\text{ctx}})$ where $e_{\text{id}}$ is a unique identifier for $e$, $e_{\text{pen}}$ is either a positive integer or $\infty$, called a penalty, $e_{\text{pi}}$ is a partial interpretation and $e_{\text{ctx}}$ is an ASP program relative to the example, called the context. A program $P$ accepts a WCDPI $e$ if and only if there is an answer set of $P \cup e_{\text{ctx}}$ that extends $e_{\text{pi}}$. In practice, $e_{\text{pen}}$ is used to bias the ILP system towards certain examples (the penalty $e_{\text{pen}}$ is paid for learning a hypothesis that does not accept example $e$), $e_{\text{pi}}$ encodes positive and negative labels as part of its inclusion and exclusion sets respectively, and $e_{\text{ctx}}$ encodes example’s contextual facts.

A context-dependent LAS task (denoted as $\text{ILP}_{\text{context}}$) is defined as a tuple $(B, S_M, E)$, where $B$ is background knowledge expressed as an ASP program, $S_M$ is the hypothesis space, defined by a language bias $M$, and $E$ is a set of WCDPIs. The hypothesis space is the set of rules that can be used to construct a solution to the task.

ILASP [15] and FastLAS [16] are two recently proposed ILP systems for solving LAS tasks. They are both capable of learning optimal solutions for $\text{ILP}_{\text{context}}$ tasks, where the notion of optimality is defined in terms of a hypothesis scoring function. The basic hypothesis scoring function, common to both systems, depends on the penalty of the examples and the length of (i.e., the number of literals in) the hypothesis. Formally, given an $\text{ILP}_{\text{context}}$ task $T_{\text{ILP}} = (B, S_M, E)$ and a hypothesis $H$:

$$S(H, E) = \sum_{e \in UNCOV(H, E)} e_{\text{pen}} + |H|$$

where $UNCOV(H, E)$ is the set of examples in $E$ that are not covered by the hypothesis $H$. A hypothesis $H \subseteq S_M$ is an optimal inductive solution of $T_{\text{ILP}}$ if and only if $S(H, E)$ is finite and there is no other hypothesis $H' \subseteq S_M$ such that $S(H', E) < S(H, E)$.

In summary, the optimisation function used by both ILP systems aims at minimising $S(H, E)$, i.e., learning a hypothesis $H$ that jointly minimises the total penalty paid for the uncovered examples and the hypothesis length. In practice, this creates a bias towards shorter, and therefore more general solutions that ignore noisy examples with a low penalty value.

3 FF-NSL Framework

FF-NSL integrates a pre-trained neural network with an ILP system for solving LAS tasks from unstructured input data. The neural network component extracts symbolic facts from unstructured input data. These facts, together with the labels of the input data, are used to automatically generate examples for the ILP

---

2 For a detailed definition of a language bias see [15].
component. The ILP component uses these examples, alongside prior background
knowledge (if any), to learn a hypothesis that predicts the labels of the input data
from the symbolic facts. At run-time, the trained FF-NSL architecture can be used
to perform a downstream classification on unseen unstructured data. An overview
of the FF-NSL architecture is presented in Figure 1, instantiated over the Sudoku
task (see Section 4.1) where input data are Sudoku grids, with digit images, la-
belled as \textit{valid} or \textit{invalid}. The novel contribution of our FF-NSL architecture is the
\textit{Example Generator} that bridges the neural and symbolic learning components.

3.1 Setup

Consider a classification task where the objective is to predict a target label \( y \in \mathbb{Z} \),
given a set of unstructured input data \( X = \{ x \in \mathbb{R}^d \} \). We denote the length of \( X \)
as \(|X|\). FF-NSL employs a \textit{pre-trained} neural network \( f_t \) to extract a discrete fact
of type \( t \) from an unstructured input data point \( x \). The neural network returns
a confidence score vector \( f_t(x) = c \in [0, 1]^k \) for \( k \) possible facts, i.e., \( f_t : \mathbb{R}^d \rightarrow [0, 1]^k \). FF-NSL takes the prediction \( s \in \mathbb{Z} \) as the fact with the maximum confidence
score from the neural network, i.e., \( s = \arg\max(c) \).

\textbf{Example 1} (Task 1: Sudoku grid validity). The objective is to predict if a Sudoku
grid \( X \), represented by a sequence of digit images \( X = \{2, 3, 4, 5, 6, 7, \ldots\} \) at
different cells in the grid is \textit{valid} (\( y = 0 \)) or \textit{invalid} (\( y = 1 \)). For each \( x_i \in X \), a pre-trained neural network extracts the symbolic fact of type \textit{digit} with the
prediction \( s_i \in \{0, 3\} \) for 4x4 grids (\( k = 4 \)), or \( s_i \in \{0, 8\} \) for 9x9 grids (\( k = 9 \)),
and an associated confidence score \( \max(c_i) \in [0, 1] \).

\textbf{Example 2} (Task 2: Follow Suit winner). The objective is to predict the winning
player \( y \in \{1, 4\} \) of a trick given by a set of four playing card images \( X = \{4, 5, 6, 7, 8, 9, 10\} \) where each card is played by a single player. The player that plays
the highest ranked card with the same suit as the card played by Player 1 is the
winner. For each \( x_i \in X \), a pre-trained neural network extracts the symbolic fact
of type \textit{card} with the prediction \( s_i \in \{0, 51\} \) (\( k = 52 \)) and an associated confidence score \( \max(c_i) \in [0, 1] \). In this task, the neural network predicts the suit and rank
of a playing card image directly.

3.2 Example Generator

The novel aspect of our FF-NSL architecture is the example generator, which
transforms unstructured input data \( X \) with corresponding label \( y \), into a WCDPI

\textsuperscript{3} We assume that a Sudoku grid has been pre-processed to return images of digits in different
cells.
digit("1, 3", 2).
digit("1, 6", 4).
digit("2, 3", 1).

(a) Sudoku grid validity

$\text{card}(1, 10, \text{hearts}),$
$\text{card}(2, \text{jack, hearts}),$
$\text{card}(3, 4, \text{clubs}),$
$\text{card}(4, 8, \text{spades}).$

(b) Follow Suit winner

Fig. 2: Example ILP contextual facts generated by FF-NSL for the Sudoku grid validity and Follow Suit winner tasks.

e is shown in Figure 2b.

Example 4 (Task 2 context generator)
Calculating the penalty of an example. In the ILP systems for learning from answer sets, WCDPI examples include a penalty used to bias the search towards hypotheses that cover certain examples. In [17], the term penalty refers to a notion of noise as a mislabelled example, in FF-NSL the penalty of an example expresses the level of “certainty” of the context of that example, which is informed by the confidence score of the neural network predictions. Given input $X$, the context $e_{ctx}$ is generated from neural network predictions $s_i = \text{argmax}(f_t(x_i))$ for every $x_i \in X$, $1 \leq i \leq |X|$. The confidence of each prediction $s_i$ is therefore quantified by $c_i = \max(f_t(x_i))$. In FF-NSL, we aggregate these individual confidence values to form the penalty $e_{pen}$ for a WCDPI example.

For our tasks, we use the minimum neural network confidence score associated with each $x_i \in X$. This is a generalisation of the binary Gödel $t$-norm used in fuzzy logics to encode fuzzy conjunctions [26,22]. Note that for both tasks, the context $e_{ctx}$ encodes a conjunction of facts for the ILP system: a conjunction of digit facts for the Sudoku task and a conjunction of card facts for the Follow Suit winner task. Experimental evaluation has shown that for the penalties to have effect they need to be sparse, hence we apply a simple linear transformation by multiplying the value by a fixed constant $\lambda > 1$. In both the Sudoku grid validity and the Follow Suit winner tasks presented in this paper, we set $\lambda = 100$ to encourage strong example coverage and also to represent sufficient variation in neural network confidence scores. Formally,

**Definition 1** (ILP Example Penalty - Generalised Gödel $t$-norm). The ILP example penalty is obtained by the function $W : [0,1]^{|X|} \rightarrow \{w \in \mathbb{Z} \mid w \geq 0\}$ defined as follows:

$$W \left( \{c_1, ..., c_{|X|}\} \right) = \lfloor \lambda \times \min\{c_1, ..., c_{|X|}\} \rfloor$$

**FF-NSL learning task.** Now we have defined how the context of an example is generated and how the associated penalty is defined, we can define the learning task for our FF-NSL framework. Let $X$ be a set of unstructured input data for one example and $y$ be an associated label from a set of labels $Y$. The example generator constructs an example $e_{\langle X,y \rangle} = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$ where $e_{id}$ is a unique example identifier, $e_{pen}$ is the example penalty $W \left( \{c_1, ..., c_{|X|}\} \right)$ generated from $X$ as defined above, $e_{pi}$ is the partial interpretation $\langle \{y\}, Y \setminus \{y\} \rangle$ and $e_{ctx}$ is the context $\{e_t(z_i, s_i) \mid 1 \leq i \leq |X|\}$ generated from the neural network predictions for each $x_i \in X$.

Given a set of labelled unstructured input data $D$ consisting of $\langle X, y \rangle$ pairs, an FF-NSL learning task is a tuple $T = \langle B, S_M, D \rangle$. An optimal solution to this task $T$ is an ASP program $H \subseteq S_M$ that is an optimal solution for the associated $\text{ILP}_{\text{context}}^\text{LAS}$ task $T_{\text{LAS}} = \langle B, S_M, E \rangle$ where $E = \{e_{\langle X,y \rangle} \mid \langle X, y \rangle \in D \}$.

**4 Evaluation**

To evaluate the proposed FF-NSL framework, we used two classification tasks, **Sudoku grid validity** and **Follow Suit winner** as introduced in Section 3.1. Examples 1 and 2 respectively. In each task, we pre-trained neural networks to extract symbolic facts from unstructured data. Then, an increasing percentage of distributional shifts were applied to input unstructured data, and with a forward
pass over the neural networks, FF-NSL generated training example contexts for an ILP system based on neural network predictions. Given corresponding labels for a downstream classification task, the ILP system learned a hypothesis which performs the final classification, based on generated contextual facts. In our evaluation, the final classification is either the validity of a Sudoku grid or the winning player of a Follow Suit card game. The evaluation considers the following:

1) Robustness to distributional shifts during learning. We demonstrate FF-NSL learns accurate and interpretable hypotheses when input unstructured data is subject to distributional shifts.

2) The effect of using an uncertainty-aware neural network. The Softmax layer, whilst commonly used to perform classification in neural networks, is known to predict with high confidence for input data subject to distributional shifts [27]. We demonstrate the robustness of FF-NSL can be improved using an uncertainty-aware neural network. We use an evidential approach [31] that leads to improved confidence estimates for out-of-distribution input data.

3) Calculating the ILP example penalty from neural network confidence scores. The novel contribution of the FF-NSL framework is creating a bias for the ILP system by setting the ILP example weight penalties based on neural network confidence scores. We quantify the effect of doing this, and, demonstrate improved performance compared to using a constant example penalty, previously used to learn from noisy examples [17].

4) Run-time performance. When data observed during training and at run-time are subject to the same proportion of distributional shift, we demonstrate FF-NSL outperforms baseline approaches trained with the same number of examples until 80% of examples are subject to distributional shift.

5) The effect of background knowledge. We demonstrate FF-NSL matches or out-performs baseline approaches when similar background knowledge is used, and background knowledge is a contributor to FF-NSL’s robustness to distributional shift.

Let us now present details of the two tasks used in our evaluation.

4.1 Sudoku Grid Validity

4.1.1 Neural networks to extract symbolic facts

We trained two types of neural networks. Firstly, we adopted the Convolutional Neural Network (CNN) architecture available in the MNIST PyTorch tutorial[4] and replaced the LogSoftmax layer with a Softmax layer and the Negative Log Likelihood loss function with Cross-Entropy Loss. This is to satisfy the neural network definition in Section 3 such that a confidence score \( c \in [0,1]^k \) is returned for \( k \) possible facts. For the two grid sizes, 4x4 and 9x9, we train two separate networks. For 4x4 grids, we set \( k = 4 \) and train on digits 1-4 inclusive, whilst for 9x9 grids we set \( k = 9 \) and train on digits 1-9 inclusive. We adopted all existing hyper-parameter values and trained for 20 epochs. When these neural networks were used with FF-NSL we use the term FF-NSL Softmax.

[4] https://github.com/pytorch/examples/tree/master/mnist
Secondly, we trained two state-of-the-art uncertainty-aware neural networks based on evidential deep learning [31], which improves the calibration of neural network confidence predictions under distributional shift. We used the available implementation in TensorFlow and set $k$, the number of outputs, to 4 and 9, for 4x4 and 9x9 grids respectively. We used existing hyper-parameter values and trained for 20 epochs. When these neural networks were used with FF-NSL we use the term FF-NSL EDL-GEN.

Confidence Score Distribution (%)

(a) 4x4 Sudoku grid validity neural networks trained with MNIST Digits 1-4 inclusive.
(b) 9x9 Sudoku grid validity neural networks trained with MNIST Digits 1-9 inclusive.

Fig. 3: Neural network performance under distributional shifts, Sudoku grid validity. Accuracy and confidence score distribution of both Softmax and EDL-GEN neural networks used in the Sudoku grid validity tasks when presented with standard MNIST digits from within the training distribution and rotated MNIST digits from outside the training distribution. Figures show the Softmax neural network predicted with high confidence for data subject to a distributional shift, despite low test set accuracy. For example, in Figure (b) when trained with standard images of MNIST digits 1-9, 55% of predictions were made with confidence in the interval [0.96, 1] for the rotated MNIST test set, despite an accuracy of 0.109. With an EDL-GEN neural network, only 3% of predictions were made with confidence in the interval [0.96, 1], showing the confidence scores of predictions made by the EDL-GEN neural network were better calibrated with its predictive accuracy.

We trained all neural networks on standard images from the MNIST training set. Figure 3 presents the test set accuracy and confidence score distribution of the trained neural networks on two test datasets; standard MNIST test digits and MNIST test digits rotated 90° clockwise, representing a distributional shift. For further dataset details, please refer to Appendix C.
4.1.2 ILP Configuration: Background Knowledge and Mode Declarations

For the Sudoku task, we used the FastLAS ILP system as FastLAS scales to large hypothesis spaces. For both 4x4 and 9x9 Sudoku grids, knowledge of the grid was encoded within the learning task presented to the ILP system. Sudoku grid cells, denoted by row and column coordinates were mapped to column, row and block identifiers. In ASP this was specified as \( \text{col}(\text{r}, \text{c}, \text{id}) \), \( \text{row}(\text{r}, \text{c}, \text{id}) \) and \( \text{block}(\text{r}, \text{c}, \text{id}) \) where \( r \) and \( c \) represent row and column coordinates, and \( id \) represents the identifier of the column, row, or block. Finally, a predicate called \( \text{neq} \) was defined to encode “not equal to" for cell identifiers.

For mode declarations, which specify the hypothesis space for the ILP system, \( \text{digit}, \text{col}, \text{row}, \text{block}, \text{and} \text{neq} \) predicates were added to the set of possible body predicates alongside negation as failure for the column, row and block predicates, i.e., \( \text{not col}, \text{not row}, \text{and} \text{not block} \). The fact \( \text{invalid} \) was added to the set of possible head atoms. The subset of the hypothesis space computed by FastLAS contained 2350 possible rules. An example listing of a Sudoku grid validity ILP task is presented in Appendix F.

For 4x4 Sudoku grids, we created an additional, more challenging learning task with a reduced set of background knowledge, where the \( \text{col}, \text{row} \text{and} \text{block} \) predicates were replaced with a \( \text{division} \) predicate that enables column, row and block identifiers to be learned, based on the cell coordinates given in the example contexts. For 9x9 grids, we also create an additional learning task for the best performing baseline approach, where neural network predictions are post-processed to create 3 Boolean input features, denoting if digits are in the same row, column or block. This effectively encodes the grid background knowledge into the learning task and goes beyond the background knowledge given to FF-NSL. We demonstrate that FF-NSL performs similarly to the best performing baseline in this case. The results for 9x9 grids, including this task, are presented in Appendix A.

4.1.3 Baselines

For all baseline approaches, we used the same pre-trained Softmax neural network as used in FF-NSL Softmax, and evaluated two alternative rule learning approaches: (1) \textit{Random Forest (RF)} which is commonly used to perform classification tasks, trains quickly and is somewhat interpretable. (2) \textit{CNN-LSTM} to evaluate a deep architecture designed for sequence classification problems where the CNN component can learn spatial dependencies in the Sudoku grid. For both the RF and CNN-LSTM, the training data consists of sequences of 16 digits (4x4 grids) and 81 digits (9x9 grids), where 0 was used to represent an empty cell and the digit values in the Sudoku grid were predicted by the FF-NSL Softmax neural networks. Each sequence was labelled with the validity of the Sudoku grid. Finally, all architecture and hyper-parameter details for the baseline approaches are presented in Appendix D.

4.1.4 Sudoku grid datasets

For each task, 10 training datasets were generated; 5 small and 5 large, each containing 320 examples and 32,000 examples respectively. Each dataset contained
an equal distribution of valid and invalid examples, and the invalid examples contained an equal distribution of examples containing two of the same digit in a row, column or block. FF-NSL was trained using small datasets and the baselines were trained with both small and large datasets. Also, two test sets were created with an additional 1000 examples, called the structured test set and the unstructured test set. The examples in the test sets were identical, except the structured test set contained structured data (e.g., digit values in the Sudoku grid) and the unstructured test set contained unstructured data (e.g., images of digits in the Sudoku grid). The structured test set was used to evaluate the accuracy of the hypothesis learned by FF-NSL, or the accuracy of the model learned using the baseline approaches. It assumes perfect predictions by the neural networks and therefore the evaluation on this dataset targets the ability of the rule learning system to handle distributional shift present during training. The unstructured test set was used for run-time evaluation, where the neural networks are required to make a prediction for digit images, which are subject to a similar proportion of distributional shift as observed during training. Let us now present the results for 4x4 Sudoku grids. Results for 9x9 grids are presented in Appendix A.

4.1.5 4 × 4 Sudoku grid validity results

Figure 4a shows the mean learned hypothesis accuracy on the structured test set over 5 repeats, where training examples were subject to an increasing percentage of distributional shift. Results for both FF-NSL Softmax and FF-NSL EDL-GEN are shown, as well as FF-NSL with reduced background knowledge, where the explicit knowledge of the Sudoku grid was removed. In this case, the EDL-GEN neural network was used. Also, Figure 4b presents the interpretability of learned hypotheses, in terms of number of atoms, assuming a hypothesis is more interpretable if it contained a lower number of atoms [14]. For the baseline RF, we obtained the learned hypothesis by inspecting the first tree in the forest and following the tree from the root down to each leaf. For the CNN-LSTM, we assumed the learned model was a black-box and applied a surrogate decision tree model to approximate model predictions [23]. We then obtained the learned hypothesis by following the decision tree from the root down to each leaf. Finally, Figure 4c presents the learning time results.

Given FF-NSL’s superior performance, we investigated why the framework is robust to distributional shift and specifically, the effect of calculating the ILP example penalty from neural network confidence scores. Firstly, we calculated the proportion of the total weight penalty in a set of generated ILP examples that was allocated to correct ILP examples. A correct ILP example is an ILP example where the generated example context and the associated label satisfy the ground truth hypothesis. The idea was to quantify how much bias was given to the ILP system, as a result of setting the ILP example penalty based on neural network confidence scores. We denote this the correct ILP example penalty ratio, calculated as
\[
\frac{\sum_{e \in E_{\text{correct}}} e_{\text{pen}}}{\sum_{e \in E} e_{\text{pen}}},
\]
where \(E\) is the set of generated ILP examples and \(E_{\text{correct}}\) is the set of correct ILP examples. For comparison, we also calculated the ratio assuming a constant penalty, where \(e_{\text{pen}} = 10\), for \(e \in E\). In this case, the ratio decreases linearly as the percentage of examples subject to distributional shift increased. Finally, we focused on the effect of this bias in terms of the accuracy of the learned
Fig. 4: Robustness to distributional shifts during learning, and the effect of background knowledge, 4x4 Sudoku grid validity. Evaluating learned hypotheses on an unseen test set containing structured data, FF-NSL outperformed the baselines until 90% of training examples were subject to distributional shift, even when the baselines were trained with 100X the number of examples and the background knowledge was reduced (Figure 4a). FF-NSL with reduced background knowledge outperforms FF-NSL with knowledge of the Sudoku grid at 90 and 100% shifts because the ILP system has more flexibility in the hypothesis space. With the grid knowledge, the hypothesis space contains rules that will perform either very well or very poorly, as the task is constrained by the grid. Hypotheses learned by FF-NSL were more interpretable, as the rules within the hypothesis contained a significantly lower number of atoms (Figure 4b). Finally, the learning time of FF-NSL with grid background knowledge was within the same order of magnitude as training the CNN-LSTM with 100X the number of examples (Figure 4c). Error bars in both Figures indicate standard error across the 5 dataset repeats.

hypothesis on the structured test set. For this analysis, we also focused on high percentages of distributional shift, {80, 90, 95, 96}%, and generated 50 datasets to ensure statistically significant results. Figure 5 presents the results for 4x4 Sudoku grids.

Finally, we evaluated FF-NSL at run-time where unseen unstructured data was observed that was subject to a similar proportion of distributional shift that was observed during training. In this case, neural networks were required to make a prediction for each digit image in a Sudoku grid and the predictions together with the learned rules were used to make a final classification. ProbLog [7], in sampling mode, was used to integrate the learned hypothesis with neural network predictions to make the final prediction, where neural network predictions were represented as annotated disjunctions, with probabilities set according to the confidence of neural network predictions. The final prediction was probabilistic, with the probabilities for a valid and invalid Sudoku grid summing to 1. We assumed the final prediction as the class with the maximum probability assigned. Our evaluation considered accuracy on the unstructured test data and also the Brier score, a scoring function designed to measure the accuracy of probabilistic predictions. The results are shown in Figure 6.
**Fig. 5:** Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, 4x4 Sudoku grid validity. Figures 5a and 5b demonstrate that calculating the ILP example weight penalty based on neural network predictions resulted in an improved bias for the ILP system, as a larger proportion of the total example weight penalty was allocated to correct ILP examples. Furthermore, the EDL-GEN neural network leads to an improvement over Softmax, as EDL-GEN was able to better estimate its predictive confidence. The Softmax neural network outperforms EDL-GEN with constant penalties because there was a greater proportion of incorrect ILP examples generated with EDL-GEN. This is shown with the 2nd and 3rd x-axes in Figure 5b. Finally, despite EDL-GEN predicting digits with slightly lower accuracy than Softmax, the calculated weight penalties when using the neural network confidence predicted by EDL-GEN created an improved bias for the ILP system and therefore outperforms Softmax, and constant penalties, in terms of learned hypothesis accuracy for FF-NSL (Figure 5c). Error bars indicate standard error across dataset repeats.

### 4.2 Follow Suit Winner

#### 4.2.1 Neural networks to extract symbolic facts

Following the Sudoku 4x4 experiments, we trained two types of neural network. Firstly, we trained a Softmax-based CNN with 4 2D convolutional layers and 2 fully connected layers for 20 epochs in PyTorch. The network accepts 3-channel RGB input with images of size 274x174 pixels and outputs a 52 dimensional softmax vector to predict each playing card. When this neural network was used within FF-NSL we use the term **FF-NSL Softmax**. Secondly, we trained an uncertainty-aware neural network based on evidential deep learning [31], modifying k, the number of outputs to 52 and the layer dimensions to accept 274x174 RGB card images. We also trained this neural network for 20 epochs, and when used within FF-NSL we use the term **FF-NSL EDL-GEN**.

We trained both neural networks on standard playing card images. Figure 7 presents the test set accuracy and confidence score distribution of the trained neu-
Fig. 6: **Run-time performance, 4x4 Sudoku grid validity**. Evaluating the FF-NSL framework on an unseen test set containing unstructured data, FF-NSL outperformed baselines trained with the same number of examples and performed similarly to baselines trained with 100X the number of examples, in terms of accuracy (Figure 6a), until 80% of the test examples were subject to distributional shift. Note that FF-NSL achieved this run-time performance whilst learning more accurate hypotheses than baseline approaches (Figure 4a). Finally, FF-NSL achieved a lower Brier score than baseline approaches trained with the same number of examples until 60% of test examples were subject to distributional shift and the Brier score of FF-NSL EDL-GEN improved as the percentage of test examples subject to distributional shift increased. This is because the EDL-GEN neural network was able to better express predictive uncertainty when data subject to distributional shift was observed, and therefore the final class probability predicted by ProbLog improved.

For the Follow Suit winner task, we used the ILASP [15] ILP system as ILASP supports predicate invention [34]. Predicate invention was required for this task as the target ground truth hypothesis included a \texttt{rank\_higher} predicate, to compare the rank value of different player’s cards, and this predicate did not exist in the context of each ILP example. For FF-NSL Softmax, we had to implement early stopping criteria that ensured ILASP returned the best scoring hypothesis after either 15 minutes had elapsed, an ILASP iteration ran for longer than 5 minutes or when ILASP achieved a candidate hypothesis score $\leq 500$. With FF-NSL EDL-GEN, no early stopping criteria was used and ILASP was allowed to find the optimal
Fig. 7: Neural network performance under distributional shifts, Follow Suit Winner. Accuracy and confidence score distribution of both Softmax and EDL-GEN neural networks used in the Follow Suit winner task when presented with standard playing card images from within the training distribution and playing card images from various card decks outside the training distribution (Batman Joker, Captain America, Adversarial Standard, Adversarial Batman Joker and Adversarial Captain America). The Softmax neural network predicted with high confidence for data subject to a distributional shift, despite low test accuracy. For example, when trained with standard playing card images, 94% of predictions were made with confidence in the interval $[0.96, 1]$ for playing card images in the Captain America test set, despite an accuracy of 0.0657. With an EDL-GEN neural network, only 9% of predictions were made with confidence in the interval $[0.96, 1]$, showing the confidence scores of predictions made by the EDL-GEN neural network were better calibrated with its predictive accuracy of 0.1193.

solution w.r.t. the scoring function. We encoded as background knowledge possible suit and rank values, the four players, as well as the definition of the rank\_higher predicate. The set of body mode declarations included a suit predicate, which linked a player’s card to a suit, alongside the rank\_higher predicate. The set of head mode declarations included a player variable, specified to support predicate invention. The hypothesis space for this task contained 96 possible rules and an example listing of a Follow Suit winner ILP task is presented in Appendix C.

4.2.3 Baselines

For all baseline approaches, we used the same pre-trained neural network as FF-NSL Softmax, and evaluated two alternative rule learning approaches: (1) Random Forest (RF) as in the Sudoku tasks. (2) Deep Fully Connected Network (FCN) to evaluate a deep neural network architecture, also commonly used to perform classification tasks. An attempt was made to evaluate a probabilistic ILP rule learner and we encountered scalability issues. We refer the reader to Section 5 for further discussion. For both the RF and FCN, the training data consisted of example
tricks, where an example contained one-hot encoded suit and numeric rank values for each playing card prediction alongside a label indicating the winner. All architecture and hyper-parameter details for the baseline approaches are presented in Section D.

4.2.4 Follow Suit winner datasets

Follow Suit winner datasets were created in a similar manner to the Sudoku datasets, as described in Section 4.1.4. 5 small and 5 large training datasets were generated, each containing 140 and 14,000 examples respectively. Also, two test sets were created with an additional 1001 examples, called the structured test set and the unstructured test set. The examples in the test sets are identical, except the structured test set contained structured data (e.g., suit and rank values of each player’s card) and the unstructured test set contained unstructured data (e.g., images of each player’s card). Further dataset details are described in Appendix C.

Let us now present the Follow Suit winner results for the Captain America deck. Full results for the Batman Joker, Adversarial Standard and Adversarial Batman Joker decks are presented in Appendix B.

4.2.5 Follow Suit winner results, Captain America

Figure 8 shows the mean learned hypothesis accuracy on the structured test set over 5 repeats, alongside interpretability and learning time. Similarly to the Sudoku experiments, for the baseline RF, we obtained the learned hypothesis by inspecting the first tree in the forest and following the tree from the root down to each leaf. For the FCN, we assumed the learned model was a black-box and applied a surrogate decision tree model to approximate model predictions [23]. We then obtained the learned hypothesis by following the decision tree from the root down to each leaf.

Also, we investigated the effect of calculating the ILP example penalty from neural network confidence scores and the effect of using an uncertainty-aware neural network. The results are shown in Figure 9 and follows the same approach as in the Sudoku experiments, outlined in Section 4.1.5. For the constant ILP example penalties, we adopted the same penalty $e_{\text{pen}} = 10$, for all $e \in E$, and focused on high percentages of distributional shift, $\{95, 96, 97, 98, 99, 100\}$.

Finally, we evaluate FF-NSL at run-time where unseen unstructured data was observed that was subject to a similar proportion of distributional shift that was observed during training. In this case, the neural networks were required to make a prediction for each player’s card image and the predictions together with the learned rules were used to make a final classification. As in the Sudoku experiments, ProbLog [2] in sampling mode, was used to integrate the learned hypothesis with neural network predictions to make the final prediction, where neural network predictions were represented as annotated disjunctions, with probabilities set according to the confidence of neural network predictions. The final prediction was probabilistic, with the probabilities for each player winning summing to 1. We assumed the final prediction as the class with the maximum probability assigned. Following Sudoku, our evaluation also considered accuracy on the unstructured test data and the Brier score. The results are presented in Figure 10.
Fig. 8: Robustness to distributional shifts during learning, Follow Suit winner, Captain America deck. Evaluating learned hypotheses on an unseen test set containing structured data, FF-NSL outperformed the baselines, even when the baselines were trained with 100X the number of examples (Figure 8a). Also, hypotheses learned by FF-NSL were more interpretable, as the rules within the hypothesis contained a significantly lower number of atoms (Figure 8b). Finally, FF-NSL’s learning time was within the same order of magnitude as the FCN trained with the same number of examples, until 40% of the training examples were subject to distributional shift with FF-NSL Softmax and 60% with FF-NSL EDL-GEN. (Figure 8c). In this task, as the EDL-GEN neural network provided significantly improved ILP example weight penalties (see Figure 9a below), ILASP required fewer iterations to learn an optimal hypothesis, in terms of minimising the total example weight penalty of examples not covered. Therefore, FF-NSL EDL-GEN learns faster, and did not require the early stopping criteria that was applied to FF-NSL Softmax, which enabled FF-NSL EDL-GEN to find the optimal solution (Figure 8a). Error bars in both Figures indicate standard error across the 5 dataset repeats.

4.2.6 Follow Suit winner detailed analysis

In this section we investigate the effect of the predictive performance of the neural networks on the percentage of incorrect ILP examples generated by FF-NSL when 95% of the training examples were subject to distributional shift. The analysis is shown in Table 1. For the Batman Joker, Captain America and Adversarial Standard decks, the EDL-GEN neural network predicted more playing cards correctly than Softmax (2nd column), which led to a lower percentage of incorrect ILP examples generated (5th column) and improved FF-NSL performance (6th column). The performance of each neural network in terms of predicting playing card ranks and suits is presented in the 3rd and 4th columns respectively.

For Adversarial Batman Joker, despite Softmax predicting more cards correctly, the predictions from EDL-GEN generated a lower percentage of incorrect ILP examples that resulted in significantly improved FF-NSL accuracy with EDL-GEN. To investigate this further, Figure 11a shows the distribution of predicted playing card rank values for both Softmax and EDL-GEN neural networks. Softmax predicted the same rank more often, with 72.70% of cards predicted
Fig. 9: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, Follow Suit winner, Captain America deck. Figures 9a and 9b demonstrate that calculating the ILP example weight penalty based on EDL-GEN neural network confidence scores resulted in an improved bias for the ILP system, as a significantly larger proportion of the total example weight penalty was allocated to correct ILP examples. On this task, the weight penalties calculated from Softmax neural network confidence scores did not provide a benefit over constant weight penalties. This is because Softmax predicted very confidently for out-of-distribution data and so the weight penalties were roughly constant (see Figure 7: 94% of Softmax predictions on the Captain America test set were made with confidence in the interval $[0, 0.96, 1]$, so data subject to a distributional shift had a similar confidence as data from within the training distribution). Therefore, the calculated ILP weight penalties were similar and provided no benefit over constant penalties for the ILP system. FF-NSL EDL-GEN with weight penalties calculated from neural network confidence scores learned a more accurate hypothesis until 100% of the training examples were subject to distributional shift (Figure 9c). Finally, the EDL-GEN neural network resulted in a lower proportion of incorrect ILP examples, as shown in the 3rd x-axis of Figure 9b. However, FF-NSL EDL-GEN with constant penalties performed similarly to Softmax in terms of accuracy (Figure 9c), so the ILP example weight penalties must have provided the benefit.

as a King, which explains the higher percentage of incorrect ILP examples. If most of the rank predictions are King, it’s unlikely the ILP example will satisfy the ground truth hypothesis (the winning player is the player with the highest ranked card with the same suit as player 1) as there won’t be a distinct player with a higher ranked card than the other players. Although EDL-GEN predicted the rank 4 for 61.57% of cards, the distribution is more varied, with higher ranked cards predicted. This results in a higher proportion of correct ILP examples. Analysing the hypotheses learned by FF-NSL, only 50% of the hypotheses learned by FF-NSL Softmax contained the correct rank higher predicate, compared to 98% with FF-NSL EDL-GEN. This closely matches the mean accuracy of the learned hypotheses in the 6th column of Table 1.
Fig. 10: Run-time performance, Follow Suit winner, Captain America deck. Evaluating the FF-NSL framework on an unseen test set containing unstructured data, FF-NSL outperformed baselines trained with the same number of examples and performed similarly to baselines trained with 100X the number of examples, in terms of accuracy (Figure 10a), until 90% of the test examples were subject to distributional shift. Note that FF-NSL achieved this run-time performance whilst learning more accurate hypotheses than baseline approaches (Figure 8a). Finally, FF-NSL achieved a lower Brier score than baseline approaches trained with the same number of examples until 70% of test examples were subject to distributional shift and the Brier score of FF-NSL EDL-GEN improved as the percentage of test examples subject to distributional shift increased. This is because the EDL-GEN neural network was able to better express predictive uncertainty when data subject to distributional shift was observed, and therefore the final class probability predicted by ProbLog improved.

For Adversarial Captain America, predictions made by Softmax led to a similar percentage of incorrect ILP examples compared to EDL-GEN (5th column in Table 1). Although this was slightly lower, the percentages for Softmax and EDL-GEN are more similar than observed in other decks). However, EDL-GEN outperformed Softmax when predicting the playing card rank. Also, looking at the distribution of playing card rank predictions, shown in Figure 11b, Softmax predicts the rank King for 58.39% of cards, whereas the EDL-GEN predicted rank distribution is significantly more varied. Analysing the hypotheses learned by FF-NSL, only 24% of the hypotheses learned by FF-NSL Softmax contained the correct rank higher predicate, compared to 70% with FF-NSL EDL-GEN. This explains the performance gap between FF-NSL Softmax and FF-NSL EDL-GEN, shown in the 6th column of Table 1.
Table 1: The effect of neural network predictive performance on the percentage of incorrect ILP examples generated by FF-NSL, and the resulting FF-NSL learned hypothesis accuracy, when using constant ILP example weight penalties. Follow Suit winner. All results shown are for mean results for 50 dataset repeats when 95% of training examples are subject to distributional shift, and the best result is highlighted in each cell.

Fig. 11: Distribution of predicted playing card rank values, Follow Suit winner, Adversarial Batman Joker and Adversarial Captain America.

5 Related Work

FF-NSL is related to work in the fields of neural-symbolic learning and reasoning, and probabilistic ILP. Within neural-symbolic integrations, there are many approaches that focus on training a neural network given a fixed logic program (neural-symbolic reasoning) [33, 21, 36]. FF-NSL is the opposite, we are given a pre-trained neural network with fixed weights and subsequently learn the logic program. In FF-NSL, the use of $t$-norms to perform aggregation over neural network predictions is similar to Real Logic [9], although FF-NSL differs in that we calculate an example weight penalty to bias the optimisation of a LAS ILP system as opposed to computing the aggregated probability of a set of probabilistic facts. $\partial$ILP [4] is an example of neural-symbolic learning and attempts to perform ILP from unstructured data, given neural network predictions, similar to FF-NSL. However, $\partial$ILP suffers scalability issues such as high memory use as a result of the generate-and-test top-down ILP approach used and the authors limit $\partial$ILP to nullary, unary and binary predicates.
Within probabilistic ILP and statistical relational artificial intelligence [6,29], systems such as ProbFOIL [5], SLIPCOVER [2], Credal-FOIL [35] and Markov Logic Networks [30] are related as a hypothesis is induced based on a set of probabilistic facts. Fundamentally, these systems adopt a different notion of uncertainty than FF-NSL and also accept different types of learning tasks. In our approach an example is either covered or uncovered (i.e., coverage is binary). In Probabilistic ILP systems, examples are covered with a probability. Probabilistic ILP systems don’t have the built-in concept of positive and negative examples and it’s impossible to ensure negative examples, represented as facts with probability 0, would not be covered by the learned hypothesis. We attempted to run the Follow Suit winner task with the Credal-FOIL system and encountered scalability issues. For the Follow Suit winner task, each example required four Credal-FOIL examples (one for each possible winner, with the target probability set to 0 for negative examples). Credal-FOIL failed to learn an accurate hypothesis (it achieved $\sim 16\%$ structured test accuracy) and took $\sim 2.5$ hours to complete, with only 50 examples.

Finally, pipeline approaches such as Concept Bottleneck Models [13] are related to FF-NSL. In particular, the independent bottleneck that performs the best on their OAI task, trains a concept extractor and higher-level learner independently, in a similar manner to FF-NSL. The models trained in [13] are all differentiable and therefore the CNN-LSTM and FCN baselines used in our tasks could be considered independent concept bottleneck models. Also, it isn’t clear if the joint and sequential bottleneck architectures would scale to the tasks presented in this paper, as multiple pieces of unstructured data are observed per example, as opposed to a single piece in [13]. Furthermore, FF-NSL could easily be extended to incorporate multiple neural networks to extract symbolic facts from different types of unstructured data for each example and therefore support a wider range of tasks.

6 Conclusion

This paper introduced a neural-symbolic learning framework called FF-NSL that learns an accurate hypothesis in the presence of unstructured training data subject to distributional shift. FF-NSL extends the LAS ILP systems and uses a pre-trained neural network to extract symbolic facts from unstructured data. Together with a label, a training example for the ILP system is generated. Neural network confidence scores are aggregated to create a penalty for each ILP example, biasing the ILP system towards learning a hypothesis that covers examples containing high confidence neural network predictions.

Our evaluation demonstrated FF-NSL was robust to distributional shift in input unstructured data observed for rule learning, outperforming random forest and deep neural network baselines. FF-NSL was able to learn a more accurate and interpretable hypothesis and required fewer training examples to do so. Also, our evaluation demonstrated the effectiveness of calculating the ILP example weight penalty from neural network confidence scores, outperforming constant example penalties. The robustness of FF-NSL was improved when using an uncertainty-aware neural network, which improved the bias for the ILP system in terms of allocating a larger proportion of the total example weight penalty to ILP examples with correct neural network predictions within the example context. On the Sudoku grid validity task, we demonstrated FF-NSL matched or outperformed the
baseline approaches when similar background knowledge was used. On the Follow Suit winner task, where ILP scalability became an issue, an uncertainty-aware neural network was able to mitigate these issues and enabled the ILP system to find an optimal solution. Finally, having learned a correct hypothesis and when unstructured test data was observed at run-time, FF-NSL was able to outperform the baseline approaches trained with the same number of examples on both tasks, until \( \sim 80\% \) of run-time test examples were subject to distributional shift.

References

1. Amodei, D., Olah, C., Steinhardt, J., Christiano, P.F., Schulman, J., Mané, D.: Concrete problems in ai safety. ArXiv abs/1606.06565 (2016)

2. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theory and Practice of Logic Programming 15 (2013). DOI 10.1017/S1471068413000689

3. Clark, K.L.: Negation as Failure, pp. 293–322. Springer US, Boston, MA (1978). DOI 10.1007/978-1-4684-3384-5_11. URL https://doi.org/10.1007/978-1-4684-3384-5_11

4. Cropper, A., Dumancic, S., Muggleton, S.H.: Turning 30: New ideas in inductive logic programming. In: Twenty-Ninth International Joint Conference on Artificial Intelligence (2020)

5. De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. In: Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI), vol. 2015-January, pp. 1835–1842. Yang, Q. IJCAI-INT JOINT CONF ARTIF INTELL (2015). URL http://ijcai.org/proceedings/2015

6. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Probabilistic Inductive Logic Programming, pp. 1–27. Springer (2008)

7. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: International Joint Conferences on Artificial Intelligence (IJCAI), pp. 2462–2467 (2007)

8. Donadello, I., Serafini, L., Garcez, A.D.: Logic tensor networks for semantic image interpretation. arXiv preprint arXiv:1705.08968 (2017)

9. Evans, R., Grefenstette, E.: Learning explanatory rules from noisy data. Journal of Artificial Intelligence Research 61, 1–64 (2018)

10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of International Logic Programming Conference and Symposium, vol. 88, pp. 1070–1080 (1988)

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

12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org

13. Kolb, P.W., Nguyen, T., Tang, Y.S., Mussmann, S., Pierson, E., Kim, B., Liang, P.: Concept bottleneck models. In: International Conference on Machine Learning, pp. 5338–5348. PMLR (2020)

14. Lakkaraju, H., Bach, S.H., Leskopec, J.: 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, pp. 1675–1684 (2016)

15. Law, M.: Inductive learning of answer set programs. Ph.D. thesis, Imperial College London (2018)

16. Law, M., Russo, A., Bertino, E., Broda, K., Lobo, J.: Fastlas: scalable inductive logic programming incorporating domain-specific optimisation criteria. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 2877–2885 (2020)

17. Law, M., Russo, A., Broda, K.: Inductive learning of answer set programs from noisy examples. Advances in Cognitive Systems (2018)

18. Law, M., Russo, A., Broda, K.: 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, pp. 196–231 (2019)
19. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
20. Lin, D., Dechter, E., Ellis, K., Tenenbaum, J.B., Muggleton, S.H.: Bias reformulation for one-shot function induction. Frontiers in Artificial Intelligence and Applications pp. 525–530 (2014)
21. Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., De Raedt, L.: Deepproblog: Neural probabilistic logic programming. In: Advances in Neural Information Processing Systems, pp. 3749–3759 (2018)
22. Metcalfe, G., Olivetti, N., Gabbay, D.M.: Proof theory for fuzzy logics, vol. 36. Springer Science & Business Media (2008)
23. Molnar, C.: Interpretable Machine Learning. Lulu.com (2020)
24. Muggleton, S.: Inductive logic programming. New Generation Computing 8(4), 295–318 (1991). DOI 10.1007/BF0337089
25. Muggleton, S., Ute, S., Alireza, T.N., Tarek, B.: Ultra-strong machine learning: comprehensibility of programs learned with ilp. Machine learning pp. 1119–1140 (2018). Special Issue of the Inductive Logic Programming (ILP) 2016
26. Ni, Q., Bertino, E., Lobo, J.: Risk-based access control systems built on fuzzy inferences. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS ’10, p. 250–260. Association for Computing Machinery, New York, NY, USA (2010). DOI 10.1145/1755688.1755719. URL https://doi.org/10.1145/1755688.1755719
27. Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J.V., Lakshminarayanan, B., Snoek, J.: Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift. In: 33rd Conference on Neural Information Processing Systems (NeurIPS) (2019)
28. Raedt, L.D., Evans, R., Muggleton, S.H., Schmid, U.: Approaches and Applications of Inductive Programming (Dagstuhl Seminar 19202). Dagstuhl Reports 9(5), 58–88 (2019)
29. Raedt, L.D., Kersting, K., Natarajan, S., Poole, D.: Statistical relational artificial intelligence: Logic, probability, and computation. Synthesis Lectures on Artificial Intelligence and Machine Learning 10(2), 1–189 (2016)
30. Richardson, M., Domingos, P.: Markov logic networks. Machine learning 62(1-2), 107–136 (2006)
31. Sensoy, M., Kaplan, L., Cerutti, F., Saleki, M.: Uncertainty-aware deep classifiers using generative models. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 5620–5627 (2020)
32. Sensoy, M., Kaplan, L., Kandemir, M.: Evidential deep learning to quantify classification uncertainty. In: Advances in Neural Information Processing Systems, pp. 3179–3189 (2018)
33. Serafini, L., Garcez, A.d.: Logic tensor networks: Deep learning and logical reasoning from data and knowledge. arXiv preprint arXiv:1606.04422 (2016)
34. Stahl, I.: Predicate invention in ilp—an overview. In: European Conference on Machine Learning, pp. 311–322. Springer (1993)
35. Tuckey, D., Broda, K., Russo, A.: Towards structure learning under the credal semantics. The 7th Workshop on Probabilistic Logic Programming, ICLP (2020)
36. Yang, Z., Ishay, A., Lee, J.: Neurasp: Embracing neural networks into answer set programming. In: C. Bossiere (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pp. 1755–1762. International Joint Conferences on Artificial Intelligence Organization (2020). DOI 10.24963/ijcai.2020/243. URL https://doi.org/10.24963/ijcai.2020/243
A 9x9 Sudoku Grid Validity Results

Let us now present the results for 9x9 Sudoku grids. We follow the same format and methodology as presented for the 4x4 Sudoku grids presented in Section 4.1.5 in the main body of the paper. Figure 12 demonstrates FF-NSL’s robustness to distributional shift during learning and that FF-NSL performs similarly to the RF baseline encoded with additional background knowledge. Figure 13 demonstrates improved performance using ILP example weight penalties calculated from neural network predictions and Figure 14 presents FF-NSL’s run-time performance.

Fig. 12: Robustness to distributional shifts during learning, and the effect of background knowledge, 9x9 Sudoku grid validity. Evaluating learned hypotheses on an unseen test set containing structured data, FF-NSL outperformed the baseline approaches, even when the baselines were trained with 100X the number of examples (Figure 12a). The RF trained with extra background knowledge performed similarly to FF-NSL. This learning task post-processed neural network predictions to create 3 Boolean input features, denoting if digits were in the same row, column or block. This effectively encoded knowledge of the Sudoku grid into the learning task and goes beyond the background knowledge given to FF-NSL, it was a significantly easier task. Hypotheses learned by FF-NSL were more interpretable, as the rules within the hypothesis contained a significantly lower number of atoms (Figure 12b). Finally, for 9x9 grids, the learning time of FF-NSL was larger than the baselines (Figure 12c). Error bars in both Figures indicate standard error across the 5 dataset repeats.
Fig. 13: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, 9x9 Sudoku grid validity. Figures 13a and 13b demonstrate that calculating the ILP example weight penalty based on neural network predictions resulted in an improved bias for the ILP system, as a larger proportion of the total example weight penalty was allocated to correct ILP examples. Figure 13c demonstrates the effect of this bias in terms of learned hypothesis accuracy (Figure 5c), where improved performance was achieved. Error bars indicate standard error across dataset repeats.

Fig. 14: Run-time performance. Evaluating the FF-NSL framework on an unseen test set containing unstructured data, FF-NSL outperformed baselines except the RF with extra knowledge, in terms of accuracy (Figure 14a). Note that FF-NSL achieved this run-time performance whilst learning more accurate hypotheses than baseline approaches (Figure 12a). Finally, FF-NSL achieved a lower Brier score than baseline approaches trained with the same number of examples until 40% of test examples were subject to distributional shift (with the exception of the RF with extra knowledge) and the Brier score of FF-NSL EDL-GEN improved as the percentage of test examples subject to distributional shift increased. This is because the EDL-GEN neural network was able to better express predictive uncertainty when data subject to distributional shift was observed, and therefore the final class probability predicted by ProbLog improved.
B Extra Follow Suit Winner Results

B.1 Batman Joker

Fig. 15: Robustness to distributional shifts during learning, Batman Joker deck.

Fig. 16: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, Batman Joker deck.
Fig. 17: Run-time performance, Batman Joker deck.
B.2 Adversarial Standard

Fig. 18: Robustness to distributional shifts during learning, Adversarial Standard deck.

Fig. 19: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, Adversarial Standard deck.
Fig. 20: Run-time performance, Adversarial Standard deck.
B.3 Adversarial Batman Joker

Fig. 21: Robustness to distributional shifts during learning, Follow Suit winner, Adversarial Batman Joker deck.

Fig. 22: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, Adversarial Batman Joker deck.
Fig. 23: Run-time performance, Adversarial Batman Joker deck.
B.4 Adversarial Captain America

Fig. 24: Robustness to distributional shifts during learning, Adversarial Captain America deck.

Fig. 25: Calculating the ILP example penalty from neural network confidence scores, and the effect of using an uncertainty-aware neural network, Adversarial Captain America deck.
Fig. 26: Run-time performance, Adversarial Captain America deck.
C Dataset Details

C.1 Sudoku Grid Validity

The Sudoku grid validity datasets were generated using valid 4x4 and 9x9 Sudoku starting configurations obtained from Hanssen’s Sudoku puzzle generator. Invalid starting configurations were obtained by taking a valid example (that didn’t exist in the set of valid examples) and changing one digit at random in a row, column or block to match another digit in the same row, column or block. All sets of invalid examples contained an equal distribution of examples containing two of the same digit in a row, column or block. The small training datasets contained 320 examples, each consisting of 160 valid starting configurations and 160 invalid starting configurations. The large training datasets contained 32,000 examples, with 16,000 valid and 16,000 invalid examples. Finally, separate test sets were created for 4x4 and 9x9 boards, which contained 1000 examples: 500 valid and 500 invalid.

For the neural network used in the 4x4 grids, we used digit classes 1-4 from the standard MNIST dataset and created a training set of 24,674 examples and a test set of 4,160 examples. The MNIST test set was further split (~70%/30%), maintaining an equal representation of digits, into two datasets as follows. The first, denoted MNIST_TEST_A contained 2910 images and was used to create FF-NSL training sets for learning a hypothesis. Digits in the Sudoku training sets were replaced with a random image of the corresponding digit from MNIST_TEST_A. The second split, denoted MNIST_TEST_B contained 1249 images and was used to create a hold out test set such that FF-NSL could be evaluated on unseen data once a hypothesis was learned. Digits in the Sudoku test set were replaced with a random image of the corresponding image from MNIST_TEST_B.

For the neural network used in the 9x9 grids, we used digit classes 1-9 from the standard MNIST dataset and created a training set of 54,078 examples and a test set of 9,021 examples. The MNIST test set was further split (~70%/30%), maintaining an equal representation of digits, into two datasets as follows. The first, denoted MNIST_TEST_A contained 6310 images and was used to create FF-NSL training sets for learning a hypothesis. Digits in the Sudoku training sets were replaced with a random image of the corresponding digit from MNIST_TEST_A. The second split, denoted MNIST_TEST_B contained 2710 images and was used to create a hold out test set such that FF-NSL could be evaluated on unseen data once a hypothesis was learned. Digits in the Sudoku test set were replaced with a random image of the corresponding image from MNIST_TEST_B.

Note that data observed by FF-NSL at learning time was completely unseen by the neural network and was therefore vulnerable to distributional shift. Also, data observed by FF-NSL at evaluation time was completely unseen by the neural network and also FF-NSL itself during learning.

Distributional shift was achieved by rotating MNIST digit images 90° clockwise in an increasing percentage of examples in the Sudoku training sets. When we evaluated with unstructured test data, the same procedure applied to the Sudoku test set, i.e., when we evaluated a hypothesis learned from a training set with 20% of the examples containing rotated images, 20% of the test set examples also contained rotated images.

C.2 Follow Suit Winner

The Follow Suit winner dataset was generated by simulating multiple games, where each game began with a randomly shuffled deck of playing cards split between the four players. Each game consisted of 13 tricks and the card played by each player along with the winner of each trick was stored. The small training datasets contained 104 example tricks from 8 games and the large training datasets contained 10,400 example tricks from 800 games. A test set was created containing 1001 example tricks from 77 games.

For the neural network, an image was taken of every playing card in a standard deck. The ImageDataGenerator class from the Keras image pre-processing library was used to apply transformations to each playing card image, generating 750 variations of each image. We set

6 https://www.menneske.no/sudoku/2
7 https://keras.io/api/preprocessing/image/
the rotation range to 55, brightness range to 0.5-1.5, shear range to 15, channel shift range to 2.5, zoom range to 0.1 and enable horizontal flip. From a total of 39,000 images, we created a training set of 27,300 images and a test set of 11,700 images (70%/30% split), maintaining an equal representation of each playing card.

Similarly to the Sudoku grid validity task, the test set was further split into two datasets (∼70%/30%), maintaining an equal representation of each playing card, as follows. The first, denoted CARDS_TEST_A contains 8164 images and was used to create FF-NSL training sets for learning a hypothesis. Playing cards in the Follow Suit winner training sets were replaced with a random image of the corresponding playing card from CARDS_TEST_A. The second split, denoted CARDS_TEST_B contained 3536 images and was used to create a hold out test set such that FF-NSL can be evaluated on unseen data once a hypothesis has been learned.

![Example playing card images](image)

Fig. 27: Example playing card images

Distributional shift was achieved by replacing playing card images from the standard deck with playing card images from alternative decks in an increasing percentage of examples in the Follow Suit winner training sets. We used playing card images from Batman Joker and Captain America decks and also created adversarial examples from each deck, placing the candidate playing card image on a background containing playing card images from the standard deck. We applied the same image transformations to the alternative decks such that standard playing card images can be directly swapped with a corresponding card image from an alternative deck. Figure 27 shows an example queen of hearts playing card image from each deck: Standard (27a), Batman Joker (27b), Captain America (27c), Adversarial Standard (27d), Adversarial Batman Joker (27e) and Adversarial Captain America (27f).

D Baseline Details

D.1 Sudoku Grid Validity

The baseline random forest model was implemented with scikit-learn 0.23.2 and tuned on the first small dataset with no examples subject to distributional shift. The number of estimators was tuned across: \{10, 20, 50, 100, 200\}. The best performing parameter value of 100 estimators was chosen and used for all Sudoku grid validity experiments. The random seed was set to 0 to enable reproducability.

The baseline CNN-LSTM consisted of an embedding layer, followed by a 1D convolutional layer with a kernel size of 3 and the ReLU activation function. Then, a 1D max pooling layer with pool size 2 was used, followed by a dropout layer, an LSTM layer and a second dropout layer. Finally, a dense fully connected layer with the sigmoid activation function was used to produce a binary classification of the input digit sequence. The input sequence length to the embedding layer was 16 for 4x4 grids and 81 for 9x9 grids, representing each cell on the Sudoku grid. We implemented the architecture in PyTorch v1.7.0.

To tune the CNN-LSTM, we sampled the learning rate \(lr \in \{0.1, 0.001, 0.0001\}\), the embedding dimension of the embedding layer \(ed \in \{32, 96, 256\}\), the number of output channels of the 1D convolution layer \(oc \in \{64, 96\}\), the number of hidden features in the LSTM layer...
\( lh \in \{32, 96, 128\} \) and the dropout probability \( dr \in \{0.01, 0.05, 0.1\} \) in both dropout layers.
We performed 10 samples and evaluated the model on the first large dataset with 0 examples subject to distributional shift, trained for 2 epochs. The best performing parameter values of \( lr = 0.0001, oc = 96, lh = 96 \) and \( dr = 0.01 \) were chosen. These parameters were then fixed for all models trained and following tuning, each model was trained for 5 epochs. Finally, the random seed was set to 0 to enable reproducability.

D.2 Follow Suit Winner

The baseline random forest model was implemented with scikit-learn 0.23.2 and tuned on the first small dataset with 0 examples subject to distributional shift. The number of estimators was tuned across: \{10, 20, 50, 100, 200\}. The best performing parameter value of 100 estimators was chosen and used for all Follow Suit winner experiments. The random seed was set to 0 to enable reproducability.

The baseline FCN consists of 3 fully connected layers with the ReLU activation function applied to each layer. Dropout was also applied after the first and second layers. Finally, a softmax layer squashed the final logits into 4 classes, representing each possible winner. The input consisted of one-hot encoded suit values and the rank value of the playing card for each player. Therefore, the input size to the first fully connected layer was 20. We implemented the architecture in PyTorch v1.7.0.

To tune the FCN, we sampled the number of output units in the first and second layers, i.e., \( l_1 \in \{20, 32, 46, 52\} \) and \( l_2 \in \{52, 64, 74, 80\} \) respectively, along with the dropout probability in both dropout layers \( dr \in \{0.1, 0.2, 0.5\} \). We sampled all possible parameter combinations and tuned on the first small dataset, with no examples subject to distributional shift, trained for 50 epochs. The best performing parameter values of \( l_1 = 20, l_2 = 74 \) and \( dr = 0.1 \) were chosen. These parameters were then fixed for all models trained and following tuning, each model was trained for 50 epochs. Finally, the random seed was set to 0 to enable reproducability.

E System Details

All experiments in this paper (with the exception of the deep neural network baselines) 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 1.1 (FastLAS 3 for 4x4 Sudoku grid validity with reduced background knowledge), ILASP 4, Python 3.7.3, PyTorch 1.7.0, TensorFlow 1.14.0, Keras 2.4.0, scikit-learn 0.23.2, numpy 1.19.1, problog 2.1.0.42. The neural network baselines were run on a machine with the following specifications:

**Hardware:** x86 compute node with 24 cores (CPU) and an NVIDIA Tesla K80 GPU, 512GB RAM.

**Operating System:** Red Hat Enterprise Linux 7.6.

**Software:** Same as above.

F Sudoku Grid Validity ILP

There are two variations of ILP tasks presented in this paper, where knowledge of the Sudoku grid was specified, and where grid knowledge was removed and replaced with a division predicate, which enabled FastLAS to learn column, row and block identifiers, based on the cell coordinates given in the example contexts. Both of these variations are presented below, with an example for 9x9 boards with the grid knowledge, and 4x4 boards without the grid knowledge. For each variation, we present the background knowledge specified, the mode declarations used and the learned hypotheses under different amounts of distributional shift. For the sudoku task with grid knowledge, we present a walk-through the FF-NSL framework, from images to learned hypothesis.
F.1 Encoding the Sudoku grid

F.1.1 Background knowledge

col("1", 1).
col("1", 2).
col("1", 3).
col("1", 4).
col("1", 5).
col("1", 6).
col("1", 7).
col("1", 8).
col("1", 9).
...
row("1", 1).
row("1", 2).
row("1", 3).
row("1", 4).
row("1", 5).
row("1", 6).
row("1", 7).
row("1", 8).
row("1", 9).
...
block("1", 1).
block("1", 2).
block("1", 3).
block("2", 1).
block("2", 2).
block("2", 3).
block("3", 1).
block("3", 2).
block("3", 3).
...

F.1.2 Mode Declarations

#modeh(invalid).
#modeb(digit(var(cell), var(num))).
#modeb(rov(var(cell), var(row))).
#modeb(col(var(cell), var(col))).
#modeb(block(var(cell), var(block))).
#modeb(neq(var(cell), var(cell))).
#maxv(4).
num(1..9).
row(1..9).
col(1..9).
block(1..9).
cell(C) :- digit(C, _).
neq(X, Y) :- cell(X), cell(Y), X != Y.
F.1.3 Example FF-NSL walk-through: 4x4 Sudoku grid validity

In this section, we present an example walk-through of the FF-NSL framework, from unstructured data to learned hypotheses, where no distributional shift was applied and where 80% of training examples were subject to distributional shift.

Let us assume a valid 4x4 Sudoku grid, shown in Figure 28, i.e., \( X = \{4, 2, 3, 7\} \) and \( y = 0 \) (valid). Let us also assume an EDL-GEN neural network, \( f_{\text{digit}} \). With no distributional shift applied, \( f_{\text{digit}} \) returned a confidence score vector for each digit image, i.e. \( f_{\text{digit}}(x_1) = [0.01, 0.04, 0.05, 0.8] \), \( f_{\text{digit}}(x_2) = [0.02, 0.05, 0.04, 0.05] \), \( f_{\text{digit}}(x_3) = [0.02, 0.05, 0.03, 0.06] \). FF-NSL EDL-GEN then generated the ILP example \( e = (1, 80, \{\text{valid}\}, \{\text{invalid}\}) \), \( \{\text{digit}(1, 2, 4), \text{digit}(2, 2, 2), \text{digit}(3, 3, 2), \text{digit}(4, 3, 1)\} \) using additional given information encoding the row and column of each digit. Given 319 additional ILP examples, the following hypothesis was learned, which states a Sudoku grid is invalid if two digits are in the same row, column or block:

\[
\text{invalid} \leftarrow \text{neq} (V_2, V_1), \ \text{digit} (V_1, V_3), \ \text{block} (V_2, V_0), \ \text{block} (V_1, V_0), \\
\text{digit} (V_2, V_3).
\]

\[
\text{invalid} \leftarrow \text{neq} (V_1, V_0), \ \text{digit} (V_0, V_2), \ \text{digit} (V_1, V_2), \ \text{row} (V_0, V_3), \ \text{row} (V_1, V_3).
\]

\[
\text{invalid} \leftarrow \text{neq} (V_1, V_0), \ \text{digit} (V_0, V_3), \ \text{digit} (V_1, V_3), \ \text{col} (V_0, V_2), \ \text{col} (V_1, V_2).
\]

When 90% of the training examples were subject to distributional shift, \( f_{\text{digit}}(x_1) = [0.2, 0.35, 0.3, 0.15] \), \( f_{\text{digit}}(x_2) = [0.2, 0.25, 0.15, 0.58] \), \( f_{\text{digit}}(x_3) = [0.3, 0.01, 0.06, 0.54] \). FF-NSL EDL-GEN generated the ILP example \( e = (1, 35, \{\text{valid}\}, \{\text{invalid}\}) \), \( \{\text{digit}(1, 2, 2), \text{digit}(2, 2, 2), \text{digit}(3, 3, 4), \text{digit}(4, 3, 4)\} \). Given 319 additional ILP examples, FF-NSL EDL-GEN learned the following incorrect rules:

\[
\text{invalid} \leftarrow \text{not} \ \text{block} (V_2, V_0), \ \text{block} (V_1, V_0), \ \text{col} (V_1, V_3), \ \text{col} (V_2, V_3).
\]

\[
\text{invalid} \leftarrow \text{not} \ \text{block} (V_2, V_0), \ \text{block} (V_1, V_0), \ \text{row} (V_1, V_3), \ \text{row} (V_2, V_3).
\]

\[
\text{invalid} \leftarrow \text{neq} (V_2, V_1), \ \text{neq} (V_2, V_3), \ \text{neq} (V_3, V_1), \ \text{block} (V_1, V_0), \ \text{block} (V_2, V_0), \ \text{block} (V_3, V_0).
\]

F.2 Without encoding the Sudoku grid

F.2.1 Background knowledge

\[
\text{div_same1}(X, Y, C) :\leftarrow (X - 1) / C = (Y - 1) / C, \ \text{idx1}(X), \ \text{idx1}(Y), \ X < Y, \ \text{quotient}(C).
\]

\[
\text{div_same2}(X, Y, C) :\leftarrow (X - 1) / C = (Y - 1) / C, \ \text{idx2}(X), \ \text{idx2}(Y), \ X < Y, \ \text{quotient}(C).
\]

\[
\text{quotient}(1..3).
\]
idx1(1..4).
idx2(1..4).

F.2.2 Mode Declarations

```prolog
# modeh (invalid).
# modeb (digit(var(idx1), var(idx2), var(num))).
# modeb (div_same1(var(idx1), var(idx1), const(quotient))).
# modeb (div_same2(var(idx2), var(idx2), const(quotient))).
# maxv(5).
num(1..4).

# bias("penalty(1, head.").
# bias("penalty(1, body(X)). :- in_body(X).").
# ground_without_replacement.
```

F.2.3 Example FastLAS learned hypotheses

With no distributional shift applied, the following rules were learned:

```prolog
invalid :- div_same1(V0, V1, 2), digit(V0, V2, V4), digit(V0, V2, V4) !=
         digit(V1, V3, V4), div_same2(V2, V3, 2), digit(V1, V3, V4), idx1(V0),
         idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- div_same1(V0, V1, 2), digit(V0, V3, V4), digit(V0, V3, V4) !=
         digit(V1, V2, V4), div_same2(V2, V3, 2), digit(V1, V2, V4), idx1(V0),
         idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- digit(V0, V2, V3), digit(V0, V2, V3) != digit(V1, V2, V3),
         (V1, V2, V3), idx1(V0), idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- digit(V0, V1, V4), digit(V0, V1, V4) != digit(V0, V3, V4),
         digit(V0, V3, V4), idx1(V0), idx2(V1), idx2(V2), idx2(V3), num(V4).
```

When 80% of training examples were subject to distributional shift, the following rules were learned:

```prolog
invalid :- div_same2(V2, V3, 2), digit(V0, V2, V4), digit(V0, V2, V4) !=
         digit(V0, V1, V4), digit(V0, V1, V4), idx1(V0), idx2(V1), idx2(V2),
         idx2(V3), num(V4).
invalid :- div_same1(V0, V1, 2), digit(V0, V3, V4), digit(V0, V3, V4) !=
         digit(V1, V2, V4), div_same2(V2, V3, 2), idx1(V0),
         idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- div_same1(V0, V1, 3), digit(V0, V3, V4), digit(V0, V3, V4) !=
         digit(V0, V3, V4), div_same2(V2, V3, 2), idx1(V0),
         idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- div_same1(V0, V1, 2), div_same1(V0, V1, 3), digit(V1, V3, V4),
         digit(V1, V3, V4) != digit(V0, V2, V4), div_same2(V2, V3, 2), digit(V0 ,V2, V4),
         idx1(V0), idx1(V1), idx2(V2), idx2(V3), num(V4).
invalid :- digit(V0, V3, V4), digit(V0, V3, V4) != digit(V0, V2, V4),
         div_same2(V2, V1, 3), div_same2(V2, V1, 3) != digit(V0, V2, V4), digit( V0, V2, V4), idx1(V0), idx2(V1), idx2(V2), idx2(V3), num(V4).
invalid :- digit(V1, V0, V3), digit(V2, V3, V4), digit(V2, V3, V4) !=
         digit(V1, V3, V4), digit(V1, V3, V4), idx1(V0), idx1(V1), idx1(V2),
         idx2(V3), num(V4).
```
G Follow Suit Winner ILP

G.1 Background Knowledge

% Suits
suit(h).
suit(s).
suit(d).
suit(c).

% Ranks
rank(a).
rank(2).
rank(3).
rank(4).
rank(5).
rank(6).
rank(7).
rank(8).
rank(9).
rank(10).
rank(j).
rank(q).
rank(k).

% Rank Value
rank_value(2, 2).
rank_value(3, 3).
rank_value(4, 4).
rank_value(5, 5).
rank_value(6, 6).
rank_value(7, 7).
rank_value(8, 8).
rank_value(9, 9).
rank_value(10, 10).
rank_value(j, 11).
rank_value(q, 12).
rank_value(k, 13).
rank_value(a, 14).

% 4 Players
player(1..4).

% Definition of higher rank
rank_higher(P1, P2) :- card(P1, R1, _), card(P2, R2, _), rank_value(R1, V1), rank_value(R2, V2), V1 > V2.

% Link player’s card to suit
suit(P1, S) :- card(P1, _, S).

G.2 Mode Declarations

P(X) :- Q(X), identity(P, Q).
P(X) :- player(X), not Q(X), inverse(P, Q).
#modem(2, inverse(target/1, invented/1)).
#modem(2, identity(target/1, invented/1)).
# predicate (target, winner/1).
# predicate (invented, pl/1).
# constant (player, 1).
# constant (player, 2).
# constant (player, 3).
# constant (player, 4).
# modeh (pl(var(player))).
# modeh (1, var(suit) != var(suit)).
# modeb (1, suit(var(player), var(suit)), (positive)).
# modeb (1, suit(const(player), var(suit)), (positive)).
# modeb (1, rank_higher(var(player), var(player)), (positive)).

G.3 Example FF-NSL walk-through: Follow Suit winner

Fig. 29: Example playing card images

In this section, we present an example walk-through of the FF-NSL framework, from unstructured data to learned hypotheses, where no distributional shift was applied and where 100% of training examples were subject to distributional shift.

Let us assume a Follow Suit winner trick, shown in Figure 29, i.e., \( X = \{\spadesuit, \\heartsuit, \diamondsuit, \clubsuit\} \) and \( y = 4 \). Let us also assume an EDL-GEN neural network, \( f_{digit} \). With no distributional shift applied, \( f_{digit} \) returned a confidence score vector for each digit image of length 52, representing a class prediction for each playing card. The EDL-GEN neural network predicted the correct class for each image respectively. FF-NSL EDL-GEN generated the ILP example \( e = \langle 1, 25, \{\text{player}(4)\}, \{\text{card}(1, 9, \text{c}), \text{card}(2, 10, \text{s}), \text{card}(3, 9, \text{c}), \text{card}(4, \text{a}, \text{c})\} \rangle \). Given 103 additional examples, FF-NSL EDL-GEN learned the following rules, which state that a player is a winner if they play the highest ranked card with the same suit as player 1:

\[\text{inverse(winner, p1)}.
\text{p1(V1)} :- \text{V2} != \text{V3}; \text{suit(V1, V2)}; \text{player(V1)}; \text{suit(V2)}; \text{suit(V3)}.
\text{p1(V1)} :- \text{rank_higher(V2, V1)}; \text{suit(V1, V3)}; \text{suit(V3, V2)}; \text{player(V1)}; \text{player(V2)}; \text{suit(V3)}.
\]

When 100% of training examples were subject to distributional shift, using the Captain America deck, FF-NSL EDL-GEN generated the ILP example \( e = \langle 1, 25, \{\text{player}(4)\}, \{\text{player}(3), \text{player}(2), \text{player}(1)\} \rangle \).
\[
\{\text{card}(1, 9, \text{c}), \text{card}(2, 10, \text{s}), \text{card}(3, \text{a}, \text{c}), \text{card}(4, \text{a}, \text{s})\}.
\]
Given 103 additional examples, FF-NSL EDL-GEN learned the following rules, which swap the players in the rank_higher predicate:

\[\text{inverse(winner, p1)}.
\]
p1(V1) :- rank_higher(V1,V2); suit(1,V3); suit(V2,V3); player(V1); player(V2); suit(V3).
p1(V1) :- V2 != V3; suit(1,V2); suit(V1,V3); player(V1); suit(V2); suit(V3).