Imperial College
London

MEng Individual Project

Imperial College London

Department of Computing

End-to-End Neurosymbolic Learning

Supervisor:
Prof. Alessandra Russo

Author:
Theo Charalambous

Second Marker:
Dr. Francesco Belardinelli

22nd June 2022

Abstract

Symbolic learning systems are a class of machine learning algorithms that aim to find a set of logical rules that explain a group of examples. Symbolic learning is often touted for the interpretability of the rules learnt, compared to neural networks which learn an incomprehensible set of weights. As machine learning algorithms are used in more critical application, symbolic learning is becoming more attractive as a pursuit since the rules learnt can be inspected and evaluated by humans.

The downside to symbolic learning algorithms is that they require input data to be encoded as a set of symbolic atoms. For tasks that take raw data inputs such as images, it is near impossible to encode the raw data symbolically. Neurosymbolic models attempt to overcome this challenge by mapping raw data to latent concepts with a neural network and then the latent concepts can be fed into a symbolic learner.

Many neurosymbolic learners are unscalable and cannot train both the neural and symbolic component jointly. This report proposes a novel ILP neuro-symbolic algorithm, NeuralFastLAS, which is able to efficiently train a neural network and symbolic learner in an end-to-end fashion. Furthermore, this report also proves the optimality of the opt-sufficient subset and demonstrates a sufficient condition on training of the neural component that guarantees the correct rules are learnt. NeuralFastLAS achieves better performance than fully neural models on tasks of arithmetic of MNIST images and reasoning about images of chess boards.

Acknowledgements

I would like to express my most sincere thanks to my supervisor, Alessandra Russo. Not only for always making time to meet with me despite her busy schedule, but for her guidance and support throughout the project and always encouraging me to push my ideas further.

Furthermore, I extend my thanks to the many people who have spent time meeting with me to discuss ideas. In particular, Yaniv Aspis, who devoted his time to join my weekly meetings with Alessandra. He provided very thorough and insightful feedback on many ideas.

I am very grateful for the chance to explore this project and truly I feel that I have had the opportunity to stand on the shoulders of giants.

Finally, I would like to dedicate this report to my grandfather. I remember spending many evenings playing backgammon with him when I was younger - he would use the opportunity to impart to me advice and knowledge that I was too young to appreciate, but his lifelong commitment to learning was something that has stuck with me.

Contents

1 Introduction **4**

1.1 Motivation 4

1.2 Contributions 4

1.3 Project Overview 5

2 Background **6**

2.1 Answer Set Programming 6

2.1.1 Constraints 7

2.2 Inductive Logic Programming 8

2.3 The FastNonOPL Pipeline 9

2.3.1 Abductive Stage 11

2.3.2 SAT-Sufficient Subset Construction 11

2.3.3 Generalisation Stage 13

2.3.4 Optimisation Stage 14

2.3.5 Solving Stage 15

3 Formulation of End-to-End Training **17**

4 NeuralFastLAS **20**

4.1 The NeuralFastLAS Pipeline 21

4.2 Abduction 21

4.3 Computing the SAT-Sufficient Subset 22

4.3.1 Symmetric Arguments 23

4.3.2 Input and Output Variables 24

4.4 Optimisation 26
4.4.1 Probabilistic Opt-Sufficiency 26
4.5 Solving Stage 29
4.5.1 Optimality of the Solving Stage 32

5 End-To-End Training with NeuralFastLAS 37
5.1 Semantic Loss in NeuralFastLAS 38
5.2 Computing the Semantic Loss 39
5.3 A Note on Rule Gradients and Learning Rate 41
5.4 Inference 41

6 Evaluation 43
6.1 MNIST Arithmetic 44
6.1.1 Learning the Posterior Rule Distribution 46
6.2 MNIST Chess 47
6.2.1 ChessMate Task 48
6.2.2 ChessState Task 49

7 Related Work 51
7.1 The $Meta_{Abd}$ Architecture 52
7.2 Semantic Loss 54

8 Conclusion 55
8.1 Future Work 55

A Ethical Considerations 62

B Evaluation Details 63
B.1 Baseline Models 63
B.2 MNIST Arithmetic Task 64
B.3 Chess Dataset 64

Chapter 1

Introduction

1.1 Motivation

Machine learning research has made huge strides in recent years, particularly in deep learning. Models like GPT-3 [1] have shown that scaling up deep models leads to improvements in performance. Despite this, it is not clear whether deep neural networks are capable of learning to reason about tasks in a logical manner - this may lead to unexpected results when generalising. The learnt parameters of a neural network are not interpretable by a human, and hence it is impossible to understand if a deep neural network has indeed learnt some neural representation of the logical rules of a task. Furthermore, deep learning models suffer many weaknesses, such as poor ability to generalise, vulnerability to bias, lack of interpretability, and poor data efficiency [2, 3].

As deep learning algorithms are used in more critical and high-risks contexts, the lack of interpretability can be problematic [4], since it is not possible to ensure that the network is learning a correct solution rather than dataset biases. Symbolic learning has the advantage of learning interpretable, logical rules, so a human can very easily understand what a symbolic learner has learnt. Difficulty arises when trying to learn symbolic rules from raw data such as images, since these are not easily encoded in a symbolic manner. Neurosymbolic architectures are used to overcome this issue, consisting both of a neural component and a symbolic component. The neural component maps the raw data to symbolic labels and the symbolic learner uses this to learn logical rules to prove the final answer.

However, in dataset where only the input raw data and final output label are provided, training both the neural and symbolic components together is a very difficult task - when untrained, the neural network passes inaccurate labels to the symbolic learner which impairs rule learning and the symbolic learner in turn back-propagates noisy gradients to the neural network. Furthermore, the symbolic component need not be differentiable, so a special mechanism is required to perform gradient descent on the neural network.

1.2 Contributions

In this report, we will explore, in detail, each stage of a novel ILP neurosymbolic architecture, NeuralFastLAS. The main contributions of this work are:

1. A novel end-to-end neurosymbolic ILP architecture, NeuralFastLAS
2. Theoretical analysis of NeuralFastLAS, in particular: proofs of the optimality of the opt-sufficient subset produced and a sufficient condition of training the neural network to guarantee the correctness of the final hypothesis.
3. A novel algorithm to learn posterior distributions on a set of possible rules with the semantic loss.
4. A discussion of the limitations the semantic loss function [5] when used in end-to-end training and proposals for potential new areas of exploration.

1.3 Project Overview

An overview of the NeuralFastLAS architecture is visualised in Figure 1.1 below. The majority of this report is dedicated to describing the NeuralFastLAS architecture - chapter 4 describes the symbolic sections of the architecture and proves that the opt-sufficient subset produced by NeuralFastLAS is optimal in that it contains an answer for the task using the ground truth latent labels. In addition, we show a sufficient condition on the accuracy of the neural network for the final solving stage to make guarantees on the correctness of the final solution. Chapter 5 discusses the neural parts of architecture and introduces a novel method of training a neural network using symbolic information when the correct rules are not yet known.

In chapter 6, we test the performance of NeuralFastLAS against fully neural models and show the advantages that NeuralFastLAS yield over fully neural models. We also discuss some of the limitations caused by usage of the semantic loss function [5] to motivate future work.

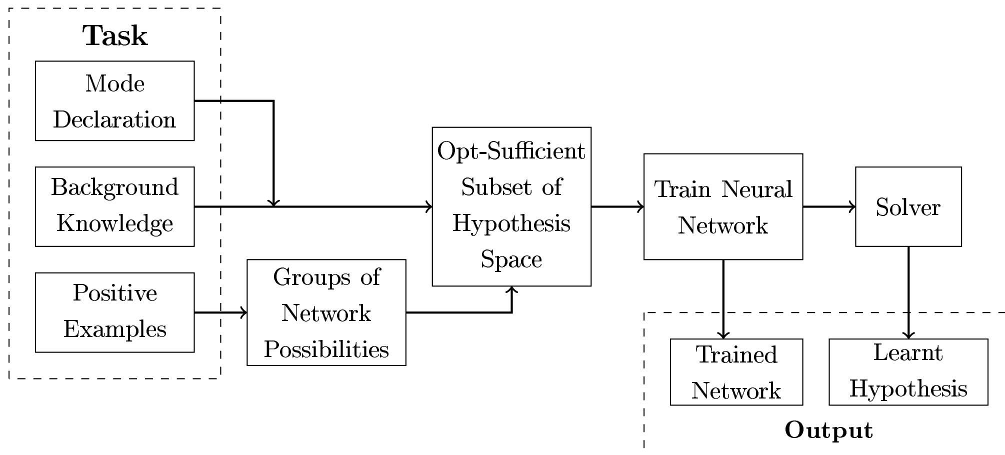

Figure 1.1: Overview of the NeuralFastLAS Algorithm

Chapter 2

Background

2.1 Answer Set Programming

Logic programming is a programming paradigm that intersects with formal logic. A logic program consists of a set of logic sentences that describe some rules in the domain of the problem. Answer set programming (ASP) is a logic programming family - another example of a logic programming family is Prolog [6].

One can build an answer set programming language from its fundamental structures: *constants* and *variables*. Constants represent objects in the domain of the language - for example, `zeus` or `athens` represent a specific God or city. On the other hand, variables are not ground, and could theoretically represent an arbitrary atom in the domain. Variables always begin with an upper case letter, whereas constants always begin with lowercase letters.

Predicates are terms that can denote meaning to a certain combination of atoms, for example `daughter(zeus, athena)` represents the fact that `athena` is the daughter of `zeus` and `brother(poseidon, zeus)` represents the fact that `poseidon` is the brother of `zeus`.

To build an understanding of ASP, we introduce a few definitions before giving some examples.

Definition 2.1.1 (Normal Rule)**.** A normal rule has the form

$$h :- b_1, ..., b_n, \text{not } c_1, ..., \text{not } c_m.$$

where h is the head atom, and the b_i and c_i are the body atoms, the c_i are all negated.

Example 2.1.2. An example of a rule in ASP would be

```
uncle(X, Y) :- brother(X, Z), parent(Y, Z).
```

which encodes the rule that `X` is the uncle of `Y` if there exists some `Z` such that `Z` is the parent of `Y` and `Z` is the brother of `X`.

Definition 2.1.3 (Subrule [7])**.** Let R be a rule, then head(R) denotes the head atom of the rule and body(R) denotes the set of body atoms in the rule. We say that R *is a subrule of* R', denoted $R \leq R'$ if $\text{head}(R) = \text{head}(R')$ and $\text{body}(R) \subseteq \text{body}(R')$. R is a strict subrule of R', denoted $R < R'$, if $R \leq R'$ and $R \neq R'$.

ASP relies heavily of the use of answer sets (or stable models [8]). Before defining a stable model, we must introduce the notion of the reduct of a logic program.

Definition 2.1.4. (Grounding of a Program [9]) Let P be a logic program. ground(P) represents the program where each rule with unground variables is replaced by the set of rules with every possible grounding.

Example 2.1.5. Consider the following program P and observe its grounding ground(P)

$$P = \begin{pmatrix} p(X) :- \text{ not } q(X), r(X). \\ q(X) :- \text{ not } p(X), r(X). \\ r(1). \\ r(2). \end{pmatrix} \quad \text{ground}(P) = \begin{pmatrix} p(1) :- \text{ not } q(1), r(1). \\ p(2) :- \text{ not } q(2), r(2). \\ q(1) :- \text{ not } p(1), r(1). \\ q(2) :- \text{ not } p(2), r(2). \\ r(1). \\ r(2). \end{pmatrix}$$

Definition 2.1.6 (Reduct [9])**.** Let P be a logic program and X a set of atoms. The reduct of P with respect to X, denoted P^X, can be constructed from P in two steps:

1. Define $P' = \{R \in P : \nexists x \in X. \text{not } x \in \text{body}(R)\}$ (i.e. deleting all rules R such that an element of X is negated in the body of R).
2. P^X is formed by removed all of the negated body atoms from the rules is P'.

Example 2.1.7. Continuing from Example 2.1.5, we now compute $(\text{ground}(P))^X$ where $X = \{p(1), q(2), r(1), r(2)\}$.

$$(\text{ground}(P))' = \begin{pmatrix} p(1) :- \text{ not } q(1), r(1). \\ q(2) :- \text{ not } p(2), r(2). \\ r(1). \\ r(2). \end{pmatrix} \quad (\text{ground}(P))^X = \begin{pmatrix} p(1) :- r(1). \\ q(2) :- r(2). \\ r(1). \\ r(2). \end{pmatrix}$$

Definition 2.1.8 (Stable Model Semantics [9])**.** An interpretation X is a stable model (or answer set) of a normal logic program P if and only if X is the least Herbrand model of $(\text{ground}(P))^X$. We denote the set of answer sets for a logic program P by AS(P)

Example 2.1.9. Again using P from Example 2.1.5, the stable models of P are given by:

$$\{r(1), r(2), q(1), q(2)\}, \{r(1), r(2), q(1), p(2)\}, \{r(1), r(2), p(1), q(2)\}, \{r(1), r(2), p(1), p(2)\}$$

2.1.1 Constraints

An ASP solver can be used to find the stable models. The ASP solver that will be used in this project is Clingo [10]. With the notion of an answer set defined, we can begin looking at the use of constraints to define which answer sets we are interested in.

Definition 2.1.10 (Hard Constraint)**.** A hard constraint is a rule with an empty head, specifically:

$$:- b_1, ..., b_n, \text{not } c_1, ..., \text{not } c_m.$$

This constraint states that there can be no answer set X with $b_1, ..., b_n, \text{not } c_1, ..., \text{not } c_m \in X$.

Hard constraints are used to eliminate answer sets, whereas there also exists weak constraints that exist to define an ordering on the answer sets.

Definition 2.1.11 (Weak Constraints)**.** A weak constraint is of the form

$$:\sim b_1, ..., b_n, \text{not } c_1, ..., \text{not } c_m.[w@l, t_1, ..., t_l]$$

where w is the weight of the constriant, l is the priority level, and the t_i are terms that define which weak constraints should be considered unique.

Whenever the body of a weak constraint is true, the term tuple is added to the cost function. If there are many penalties with a variety of priorities, then the answer sets are compared by lexicographically comparing the sum of the costs over each priority (with the 0-th priority being the first priority). If weak constraints are present in an ASP program, Clingo will attempt to find the answer set such that the cost function is minimal.

2.2 Inductive Logic Programming

Inductive logic programming is a form of machine learning. It aims to learn a set of hypothesis (rules) H that allow the proof of positive examples from a background knowledge.

Two common systems for ILP tasks are ILASP [11] and FastLAS [7]. This project primarily relates to FastLAS, so the definitions in this section will be geared towards understanding the FastLAS algorithm.

More formally, let the background knowledge B be a set of clauses (possible empty), define the positive examples E^+ and negative examples E^- to each be a set of atoms. We now introduce the necessary concepts to define a learning from answer sets task.

Definition 2.2.1 (Partial Interpretation [12])**.** A partial interpretation e_{pi} is a pair of disjoint sets of atoms $\langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle$ called the inclusions and exclusions respectively.

We say that an interpretation X extends e_{pi} if $e_{\text{pi}}^{\text{inc}} \subseteq X$ and $e_{\text{pi}}^{\text{exc}} \cap X = \varnothing$.

Definition 2.2.2 (Weighted Content-Dependant Partial Interpretation [12])**.** A weighted context-dependant partial interpretation (WCDPI) is a tuple $e = \langle e_{\text{id}}, e_{\text{pen}}, e_{\text{pi}}, e_{\text{ctx}} \rangle$ where e_{id} is a string identifier for the example, e_{pen} is a penalty which is either a positive integer or ∞, e_{pi} is a partial interpretation and e_{ctx} is a program consisting of normal rules.

Definition 2.2.3 (Mode Bias [7])**.** The structure $M = \langle M_h, M_b \rangle$ defines the mode bias of the hypothesis space. M_h and M_b are sets of literals with abstracted arguments of the form `var(t)` or `const(t)`. A literal l is said to be compatible with a mode declaration m if l can be constructed from the mode declaration by replacing all of the `var(t)` in m with variables of type `t` and replacing the `const(t)` with ground terms of type `t`.

A set of rules $\{H_i\}_i$ is present in the hypothesis space if for each H_i, head(H_i) is compatible with an element in M_h and every term in body(H_i) is compatible with an element in M_b.

The search space S_M is the set of all hypotheses that are compatible with the mode bias M.

Definition 2.2.4 (Learning from Answer Set Task [12])**.** A positive learning from answer sets task is a tuple $T = \langle B, M, E^+ \rangle$ where B is the background knowledge, M is the mode bias, and E^+ is a finite set of WCDPIs.

We say that a hypothesis H covers an example $e \in E^+$ if $B \cup H \cup e_{\text{ctx}}$ extends e_{pi}. The score of a hypothesis H is given by

$$\text{score}(H) = |H| + \sum_{\substack{e \in E^+ \\ H \text{ does not cover } e}} e_{\text{pen}}$$

and we say a hypothesis H is optimal if score(H) is finite and there does not exist any H' such that score$(H') <$ score(H).

Remark 2.2.5. It is interesting to note that the background knowledge is provided by the user, in this way ILP differs from most common machine learning tasks such as neural networks and reinforcement learning by allowing the user to provide some domain specific knowledge to the learning agent to aid learning.

2.3 The FastNonOPL Pipeline

In this project, majority of the learning tasks are non-observational. FastLAS2[1] [12] is used in this project. Many stages of the FastNonOPL pipeline are edited to introduce new functionality in NeuralFastLAS, so we dedicate a section of this chapter to summarising each stage of the pipeline as described in [12].

In order to motivate this summary, we introduce the concept of a non-observational predicate learning task alongside an example.

Definition 2.3.1 (Non-Observation Predicate Learning Task [12])**.** A task $\langle B, M, E^+ \rangle$ is non-observational if the search space contains a rule whose head uses a predicate whichs occurs in the body of a rule in B or in the body of a rule in context of an example in E^+.

Example 2.3.2 (A Non-OPL Task)**.** Consider a situation where a university wishes to automate keycard creation for the members of the university. To do this, the university must learn which traits are require for a person to access specific rooms. Suppose we have the following set-up:

	Traits			Can access		
	Professor	Has lab training	Visitor	Lab	Staff Room	Storage Room
Daisy	✓	✓		✓	✓	✓
Chris	✓				✓	
Bella		✓		✓		
Adam			✓			

The background knowledge is given by:

```
can_access(Person, Room) :- not visitor(Person),
                            card_opens(Person, Room).
```

Then the solution is given by:

```
card_opens(Person, lab)          :- has_lab_training(Person).
card_opens(Person, staff_room)   :- is_professor(Person).
card_opens(Person, storage_room) :- is_professor(Person),
                                    has_lab_training(Person).
```

[1]The code for the FastLAS2 algorithm can be found at https://github.com/spike-imperial/FastLAS

This is a non-observational task (albeit a simple one), since the head of the rules being learnt, namely `card_opens/2`, does not appear as a label - the labels are of the predicate `can_access/2`.

Note that the FastLAS algorithm can only solve a subset of ILP tasks, specifically it cannot solve recursive learning tasks, which we formalise in the following definition.

Definition 2.3.3 (FastLAS Partitioning Condition [12]). An ILP task $T = \langle B, M, E^+ \rangle$ is non-recursive if for each $e \in E^+$, $B \cup e_{\text{ctx}}$ can be partitioned into two disjoint programs - bottom$(B \cup e_{\text{ctx}})$ and top$(B \cup e_{\text{ctx}})$ - such that no predicate in M_h or the head of a rule in top$(B \cup e_{\text{ctx}})$ occurs in M_b or the body of a rule in bottom$(B \cup e_{\text{ctx}})$.

Example 2.3.4. An example of a recursive program may be with where the desired rule is the familial relation between ancestors:

```
ancestor(P1, P2) :- ancestor(P1, P3), parent(P3, P2).
```

This is clearly recursive since `ancestor/2` $\in M_h$ and `ancestor/2` $\in M_b$.

The FastLAS algorithm also has a specific notion of a scoring function formalised in the definition below.

Definition 2.3.5 (Scoring Function [7]). Let $\mathcal{T}_{\text{nopl}}$ and $\mathcal{P}$ represent the set of all possible non-observational learning tasks and the set of all logic programs respectively.

A scoring function is a function $S : \mathcal{P} \times \mathcal{T}_{\text{nopl}} \to \mathbb{R}_{\geq 0}$. A hypothesis H is said to be optimal with respect to S if there does not exist any H' such that $S(H', T) < S(H, T)$.

Finally, for a scoring function S, S^{rule} is said to be a decomposition of S if for all $P \in \mathcal{P}, T \in \mathcal{T}_{\text{nopl}}$, $S(P, T) = \sum_{R \in P} S^{\text{rule}}(R, T)$.

FastLAS requires the score function to be of the form

$$S(H, T) = S'(H, T) + \sum_{e \in \text{Uncov}(H,T)} e_{\text{pen}}$$

where $S'(H, T)$ is decomposable. An example of a decomposable scoring function often used is simply $S'(H, T) = |H|$ (the number of literals in H).

Now that we have defined the fundamental concepts used within FastLAS, we can begin to explore each of the stages in detail. The non-observational version of FastLAS have N stages:

1. **Abduction:** Transform the non-observational task into an observation one by finding all abductive explanations for a label in terms of the non-observation predicates in M_h.
2. **Generate SAT-Sufficient Subset:** Generate a subset $C^+(T)$ of the hypothesis space S_M which is guaranteed to contain at least one optimal solution.
3. **Generalisation:** Search for rules that are subrules of multiple rules $C^+(T)$ and cover multiple examples.
4. **Optimisation:** For each rule in the generalised set, compute an optimal subrule with respect to the scoring function.
5. **Solving:** Find an optimal hypothesis from the set of optimised rules.

2.3.1 Abductive Stage

The first stage of FastNonOPL involves turning the non-observational task into an observational one via abduction. For each $e \in E^+$, possibilities are built iteratively. Let e be a CDPI of the form $\langle\langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}}\rangle, e_{\text{ctx}}\rangle$, the starting set of partial possibilities is of the form $\{\langle\langle\varnothing,\varnothing\rangle, A\rangle : A \in \text{AS}(B \cup e_{\text{ctx}})\}$.

In each iteration of the abductive stage, the algorithm performs "anti-abduction" [12] to search for exceptions to the CDPIs found in the previous iteration.

Definition 2.3.6 (Exception of a possibility)**.** An exception of a possibility $\langle\langle p_{\text{pi}}^{\text{inc}}, p_{\text{pi}}^{\text{exc}}\rangle, A\rangle$ is a set Δ of literals such that Δ extends p_{pi} but there does not exist any answer set $\Delta \cup A \cup \text{top}(B \cup e_{\text{ctx}})$ that extends e_{pi}.

An exception is said to be minimal if there does not exist any exceptions of a smaller size.

In each iteration, after identifying the exceptions, the algorithm then attempts to resolve these exceptions via positive or negative fixes.

- A positive fix consists of resolving an individual minimal exception by extending $p_{\text{pi}}^{\text{inc}}$ such that Δ no longer extends p_{pi} but p_{pi} still consists only of atoms which occur in the head of some rule in the search space.
- A negative fix extends $p_{\text{pi}}^{\text{exc}}$ with all of the minimal exceptions. The resulting possibility is guaranteed to be exception-free.

The algorithm continues iteratively constructing possibilities until there are no more exceptions.

Definition 2.3.7 (Possibility [12])**.** Possibilities are of the form $\langle p_{\text{pi}}, p_{\text{as}}\rangle$, where p_{pi} is a partial interpretation, and one may inutuitively understand p_{as} to correspond roughly to an answer set of the bottom program bottom$(B \cup e_{\text{ctx}})$.

Remark 2.3.8. An important point to emphasise is that by building up the possibilities of each example e from $\text{AS}(B \cup e_{\text{ctx}})$ means that FastNonOPL supports choice rules and hard constraints in the background.

2.3.2 SAT-Sufficient Subset Construction

The rest of the FastNonOPL pipeline is the same as the original FastLAS pipeline [7] since the abduction stage produces an OPL task, however there are small differences due to the use of possibilities which we discuss in the later stages.

Definition 2.3.9 (Characteristic Ruleset [7])**.** Let $e \in E^+$, $p \in$ possibilities(e) and a be a ground atom. A rule R is in the characteristic ruleset of T with respect to a and p, denoted by $C(T, a, p)$ if the following hold:

1. $R \in S_M$
2. There is at least one ground instance R^g of R such that head$(R^g) = a$ and body(R^g) is satisfied by the unique answer set of p_{as}.

3. There is no rule R' that satisfies the first two conditions such that R s a strict subrule of R'

Note that the final condition of the elements in $C(T, a, p)$ forces the rules to be as specific as possible with respect to the search space. This means that these rules can be rather long.

However, we are not specifically interested in the characteristic rulesets with respect to specific examples and atoms, but instead the union of all of these rulesets with respect to the inclusions and exclusions.

Definition 2.3.10 (Characterisation of T)**.** We define the positive characterisation of a possibility p in a task T, denoted $C^+(T, p)$, as

$$C^+(T, p) = \bigcup_{a \in p_{\text{pi}}^{\text{inc}}} C(T, a, p)$$

and the negative characterisation of a possibility p in a task T, denoted $C^-(T, p)$, as

$$C^-(T, p) = \bigcup_{a \in p_{\text{pi}}^{\text{exc}}} C(T, a, p)$$

With this, we can define the positive and negative characterisation of the task, denoted $C^+(T)$ and $C^-(T)$ respectively, as

$$C^+(T) = \bigcup_{e \in E^+} C^+(T, e) \qquad C^-(T) = \bigcup_{e \in E^+} C^-(T, e)$$

where $C^{\pm}(T, e)$ is simply the union of $C^{\pm}(T, p)$ for every $p \in \text{possibilities}(e)$.

To emphasise the importance of the SAT-sufficient subset, we recall Theorem 1 from [7]:

Theorem 2.3.11 (Theorem 1 from [7])**.** T is satisfiable if and only if $C^+(T)$ contains an inductive solution of T.

C^+ is SAT-sufficient since by construction C^+ is itself a solution of T - that this is why it is necessary for the rules in $C(T, a, p)$ to be very specific.

The characteristic rules are computed using a meta-level ASP encoding of the task in Clingo [10]. The rulesets for each example can be computed independently from one other.

Example 2.3.12. As an example of an meta-level ASP program that the SAT sufficient stage use, consider a learning task T with an example e with identifier `eg1`. Each atom `a` in p_{as} is present in the program P in the form `ctx(eg1, a)`. The inclusion and exclusion sets are encoded with `example_inclusion/2` and `example_exclusion/2` respectively, where the first argument denotes example identifier, and the second argument is a literal from e_{pi}.

To see how the mode bias is encoded in the meta-level ASP program, suppose first that `f(var(number), var(number))` is in M_h. This will generate the rule

```
possible_head(f(ARG0, ARG1)) :- target(_, _, f(ARG_VAL0, ARG_VAL1)),
                                var(ARG0), var(ARG1),
                                ctx(EG, number(ARG_VAL0)),
                                ctx(EG, number(ARG_VAL1)).
```

where `target/3` is used to encode the literal from p_{pi} that the answer set corresponds to. The first argument is the example identifier, the second is either `inc` or `exc` (to encode whether the literal is from $p_{\text{pi}}^{\text{inc}}$ or $p_{\text{pi}}^{\text{exc}}$) and the last argument is the literal. Note that there is only one target per answer set (this is controlled by a choice rule).

Since each rule only has one head atom, the program makes a choice over all possible valid head atoms with the rule

```
1 { head(H) : possible_head(H), not invalid_head(H) } 1.
```

Body conditions are encoded slightly different since there is no constraints on the number of positive body literals. Suppose now that `g(var(number), var(number))` is in M_b, then the following rule is generated:

```
{ in(g(ARG0, ARG1)) } :- ctx(EG, g(ARG_VAL0, ARG_VAL1)),
                         var(ARG0), var(ARG1),
                         ctx(EG, number(ARG_VAL0)),
                         ctx(EG, number(ARG_VAL1)).
```

We can read this as if a grounded form of the literal is in p_{as}, then we can include the positive literal in the body of the rule.

When it comes to solving, we can create a 1-1 map between elements of $C(T, a, p)$ and answer sets of the Clingo program. To do this, consider one answer set of the program, then let a correspond to `A` in `target(_, _, A)`. The element of $C(T, a, p)$ being mapped to can be built with the arguments of `head/1` and `in/1`. For example, suppose the following was an answer set:

```
head(f(v_a_r0, v_a_r1)) in(g(v_a_r1, v_a_r1)) in(h(v_a_r0, v_a_r1, v_a_r2))
```

This corresponds to the rule

```
f(V0, V1) :- g(V1, V1), h(V0, V1, V2).
```

2.3.3 Generalisation Stage

The next step is to try and generalise the highly specific rules in $C^{+}(T)$. To do this, we define for following notion of a generalisation set:

Definition 2.3.13 (Generalisation of Characterisation)**.** For a rule R, we define

$$c_R = \{R' : R' \in C^{+}(T), R \leq R'\}$$

where $R \leq R'$ denotes that R is a subrule of R'. With this, we can define the generalisation set as

$$G(T) = \{R : R \in S_M, c_R \neq \varnothing, \nexists R' : [R < R' \wedge c_R = c_{R'}]\}$$

The final condition on the elements of $G(T)$ ensures that if two rules R_1 and R_2 are generalisation of the exact same rules in $C^{+}(T)$, only the most general rule is contained within $C^{+}(T)$.

In FastLAS, the $G(T)$ set is computed by looking at every rule in $C^{+}(T)$ and checking every possible subrule to check if it is in $G(T)$. This is done in C++.

2.3.4 Optimisation Stage

The generalisation stage only generalises rules if it is possible to generalise multiple rules from C^+ with the generalisation. It is also necessary to generalise rules with respect to the scoring function and the $C^-(T)$ set. To do this, we introduce the following notion of optimisation:

Definition 2.3.14 (Rule Optimisation [7])**.** Fix a rule $R \in S_M$ and decomposable score function S, we say that R' is an optimisation of R if:

1. $R' < R$
2. There does not exist any $e \in E^+$ such that $e_{\text{pen}} = \infty$ and R' is a subrule of some rule in $C^-(T, e)$.
3. There does not exist any other rule R'' such that $S^{\text{rule}}(R'', T) < S^{\text{rule}}(R', T)$.

R is said to be optimisable if at least one optimisation exists.

Definition 2.3.15 (Optimised Characteristic Hypotheses Space [7])**.** The optimisation step produces an optimised characteristic hypotheses space of T with respect to S which is defined to be a set containing at least one optimisation of each optimisable rule in $G(T)$.

The optimised characteristic hypothesis space is computed in FastLAS via an ASP encoding. Since the scoring function is decomposable, each rule can be optimised independently from one another.

The scoring function S^{rule} must be expressed via ASP using the predicate `penalty/2`, where the first argument represents the value of the penalty and the second value is a term that matches on the literals of a hypothesis (head literals of the hypothesis are of the form `in_head(Literal)` and body literals are of the form `in_body(Literal)`)

Example 2.3.16. The function $S^{\text{rule}}(H, T) = |H|$ would be encoded as

```
penalty(1, head(X)) :- in_head(X).
penalty(1, body(X)) :- in_body(X).
```

The scoring function $S^{\text{rule}}(H, T) = |\text{head}(H)| + 5|\text{body}(H)|$ would be encoded as

```
penalty(1, head(X)) :- in_head(X).
penalty(5, body(X)) :- in_body(X).
```

The identifiers `head(X)` and `body(X)` are necessary in the second argument of `penalty/2` so that the penalty is paid for each literal. To highlight this point, consider the encoding

```
penalty(1, head) :- in_head(X).
penalty(1, body) :- in_body(X).
```

Clingo answer sets do not contain repeated atoms, so this rule would only contain at most one penalty term for the head and body. While this isn't a problem for the head, since the rules generated by FastLAS only contain one head atom, the body of any rule would only generate a penalty of 0 or 1 regardless of how many literals are actually in the body.

With this penalty term, Clingo can optimize the answer sets with respect to the scoring function by including the following weak constraint

```
:~ penalty(P, ARGS).[P@0, ARGS]
```

With this, Clingo will attempt to find the smallest answer set with regards to the sum of penalty terms in each answer set. The encoding defines a choice of whether to include each body literal, so the answer sets will correspond to a subrule of the rule being optimised.

The final output from this stage will be a optimised characteristic hypotheses space of T - specifically, a set containing at least one optimisation for each optimisable rule in $G(T)$. We will denote this set of $O(T)$. By construction, $O(T)$ is opt-sufficient with respect to the scoring function, meaning that either T is unsatisfiable or $O(T)$ contains at least one optimal solution of the task. Details of the proofs of this claim can be found in [7].

2.3.5 Solving Stage

Once $O(T)$ has been constructed, the final solving stage can commence. The solving stage finds the optimal solution with respect to the scoring function from $O(T)$. Since FastLAS has constraints on the type of tasks it can solve, it can solve the task by solving a single ASP encoding of the task, which we will call P_{solve}.

To construct P_{solve}, we have the following elements [13]:

1. For each hypothesis $\mathtt{h} \in O(T)$, P_{solve} contains the following choice rule and weak constraint:

   $\mathtt{0\ \{\ in_h(h_{id})\ \}\ 1.}$
   $\mathtt{:\sim\ in_h(h_{id}).[}S^{rule}(H,T)\mathtt{@1,\ in_h(h_{id})].}$

2. For each example $\mathtt{e} \in E^{+}$, if e has an infinite penalty, P_{solve} contains the hard constraint

   $\mathtt{:-\ not\ covered(e_{id}).}$

   Otherwise the penalty is finite, and P_{solve} contains the weak constraint

   $\mathtt{:\sim\ not\ covered(e_{id}).[e_{pen}@1,\ eg_id(e_{id})]}$

   In FastNonOPL, an example is covered if at least one of its possibilities are covered, so for each possibility $\mathtt{p} \in \text{possibilities}(\mathtt{e})$, we add the rule

   $\mathtt{covered(e_{id})\ :-\ not\ poss_not_covered(p_{id}).}$

   For a given possibility, this possibility is covered by a hypothesis $H \in O(T)$ if:

   (a) For every $a \in \mathtt{p}_{\text{pi}}^{\text{inc}}$, there exists at least one rule $H' \in C^{+}(T, a, \mathtt{p})$ such that $H \leq H'$.
   (b) For every $\hat{a} \in \mathtt{p}_{\text{pi}}^{\text{exc}}$, there exists no rule $H' \in C^{+}(T, a, \mathtt{p})$ such that $H \leq H'$

   For purposes of expressiveness, we define

   $$O^{+}(T, a, \mathtt{p}) = \{H : H \in O(T), \exists R \in C^{+}(T, a, \mathtt{p}).\ H \leq R\}$$
   $$O^{-}(T, a, \mathtt{p}) = \{H : H \in O(T), \exists R \in C^{-}(T, a, \mathtt{p}).\ H \leq R\}$$

   Using this, we can encode these two statements in ASP. For each $\mathtt{a} \in \mathtt{p}_{\text{pi}}^{\text{inc}}$, we add the rule

$$\texttt{poss_not_covered(p}_{\texttt{id}}\texttt{) :- } \bigwedge_{\mathtt{h} \in O^{+}(T,\mathtt{a},\mathtt{p})} \texttt{ not in_h(h}_{\texttt{id}}\texttt{).}$$

and for each $\mathtt{a} \in \mathtt{p}_{\mathrm{pi}}^{\mathrm{exc}}$ and $\mathtt{h} \in O^{-}(T, \mathtt{a}, \mathtt{p})$, we add the rule

$$\texttt{poss_not_covered(p}_{\texttt{id}}\texttt{) :- in_h(h}_{\texttt{id}}\texttt{).}$$

With the program P_{solve} defined, we can use Clingo to find the answer set A that minimizes the penalities. From this answer set, we can build the optimal solution $H \subseteq O$ by including h in H if $\texttt{in_h(h}_{\texttt{id}}\texttt{)} \in A$.

Chapter 3

Formulation of End-to-End Training

End-to-end training of neurosymbolic models is a very novel area of research and hence there exists vast opportunities for exploration available to us. This section formally introduces the notion of end-to-end training.

A neurosymbolic model generally has the following form: a neural perceptron model and a symbolic reasoning model [14]. Consider a dataset of the form $\mathcal{D} = \{\langle x_i, y_i \rangle\}$. The neural component of the model maps the input $x_i \in \mathcal{X}$ to an element z_i in the latent space $\mathcal{Z}$ which represents the symbolic inputs the symbolic reasoning component. The symbolic component then uses the latent concept z_i to solve some logic program P_i to find the final answer of the problem y_i. y_i could represent a set of classes in the case of multi-label classification, but in single-label classification, y_i is a single class.

In general, the input x_i may consist of multiple raw data inputs (for example, a group of MNIST digits). In this case, each element of $\mathcal{X}$ is an n-tuple of raw data inputs. In this case, the elements of $\mathcal{Z}$ will also be n-tuples.

It is important to note that the labels for latent space $\mathcal{Z}$ are not present in the dataset annotations in any form. It is up to the definition of the model schematic to specify the latent space and create a perceptron model that will map inputs to the latent space.

Definition 3.0.1 (Neural Network Parameterisation)**.** The neural component is parameterised by θ and the outputs are representated with the conditional probability $P_\theta(z|x)$, which is read as the probability that the neural network outputs z with inputs x and parameters θ.

The space of all possible network parameters is denoted Θ.

While not normally explicitly stated, in the report we require the constraint, for a given $x \in \mathcal{X}$ and $\theta \in \Theta$

$$\sum_{z \in \mathcal{Z}} P_\theta(z|x) = 1$$

Example 3.0.2. To give a concrete example, we run through a simple task. Let the dataset in this example be of the form $\mathcal{D} = \{\langle \langle x_{i,1}, x_{i,2}, x_{i,3} \rangle, y_i \rangle\}_i$ where each $x_{i,j}$ is a randomly sampled MNIST [15] image and y_i is the sum of the MNIST digit values of $x_{i,1}, x_{i,2}$ and $x_{i,3}$. Let X denote the set of MNIST digits, then the input space for this task is $\mathcal{X} = X \times X \times X$ and the output space is $\mathcal{Y} = \{0, 1, ..., 27\}$.

In this case, the most reasonable latent space for the dataset is set set of all possible MNIST digit values, so more specifically it is the cartesian product $\mathcal{Z} = \mathbb{Z}_{10} \times \mathbb{Z}_{10} \times \mathbb{Z}_{10}$.

The neural component of the model will be responsible for mapping the raw images to their corresponding numerical value, and the symbolic component will be responsible for mapping the numerical values to their sum. Figure 3.1 shows an example of the forward pass of such a trained model.

In this example, the neural component may be a trained convolutional neural network and the symbolic component can be a rather simple ASP program. Given $\langle z_{i,1}, z_{i,2}, z_{i,3} \rangle$, we can produce the program P_i:

```
nn(1, z_{i,1}). nn(2, z_{i,2}). nn(3, z_{i,3}).
answer(Y) :- nn(1, Z1), nn(2, Z2), nn(3, Z3),
             Y = Z1 + Z2 + Z3.
```

Note that only the first line of the program is changed between each P_i to encode the different outputs from the neural component for each datapoint. Running each P_i with Clingo and looking at the argument of `answer/1` will result in the final prediction from the label.

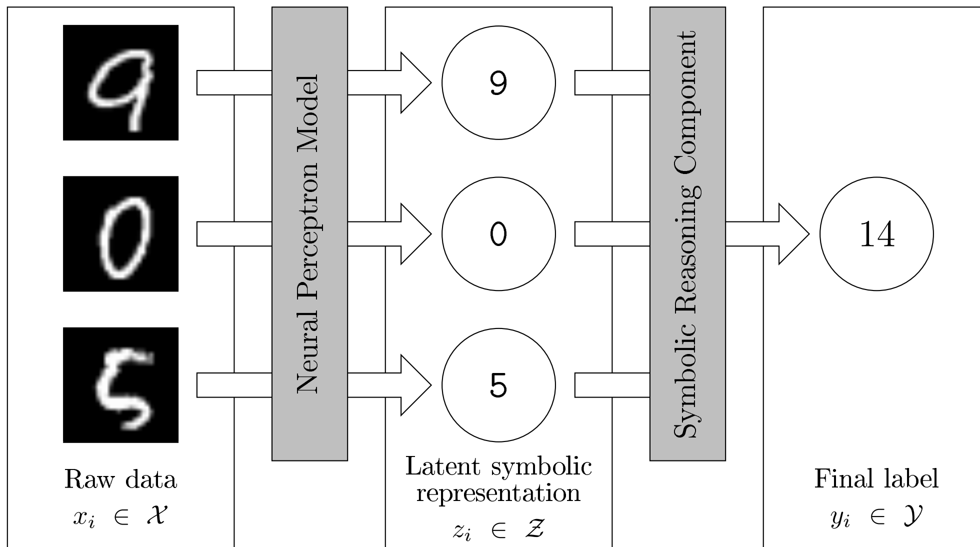

Figure 3.1: Example of the form of a neurosymbolic model

The example above assumes that the model is already trained. However both components have to be trained in either an isolated or end-to-end form:

1. **Training the Perceptron Component:** If the perceptron component is a neural network, it is of course necessary to train it in some way to predict the correct latent symbolic representation.
   - **Isolated Training:** If the raw data is from some dataset that labels each datapoint (such as in Example 3.0.2, the raw data is from the MNIST dataset, so the network could be pretrained on the MNIST dataset, then the pretrained model can be used in the neurosymbolic architecture). The isolated training method only works if there exists labels for the latent space $\mathcal{Z}$, so it is not sufficiently general to rely on as a method.
   - **End-to-End Training:** End-to-end training requires some form of loss to be propogated from the symbolic component back to the neural component. This is, in general, a non-trivial task since the symbolic component need not be differentiable.

2. **Training the Symbolic Component:** The symbolic component can also be trained by learning the correct rules. The problems in this case is that if the model is trained end-to-end, the predictions from the neural network may not be sufficiently accurate for the symbolic component to be able to find the optimal hypothesis H^* (or possibly even any hypothesis).

   In Example 3.0.2, the rule was given, stating that the answer was the addition of the three latent labels. However, one can imagine that there exists more complex datasets where either the rules are too complex to engineer manually or the latent space is not easy to comprehend and so better rules will be created by learning rather than engineering.

The complexity of both training the neural component without true supervision and training the symbolic component with inaccurate input labels makes the goal of end-to-end training very difficult and may explain why it is a relatively unexplored area of research.

Chapter 4

NeuralFastLAS

Having defined the goal of an end-to-end architecture in the previous chapter, we now introduce a novel ILP algorithm that incorporates probabilistic facts from a neural network within the symbolic algorithm to train a neurosymbolic model in an end-to-end manner. We can see this as empowering symbolic rule learning with the ability to extract latent symbolic concepts from raw data while learning general knowledge.

The algorithm is based on the FastLAS architecture [7], the architecture is built upon with many stages of the original algorithm being changed to be compatible with the neural predictions and a new probabilistic optimization criteria.

Definition 4.0.1 (NeuralFastLAS Task)**.** A NeuralFastLAS task is of the form $T = \langle B, M, E^*, \mathcal{Z} \rangle$ where $\mathcal{Z}$ is the latent space and E^* is a set of probabilistic examples where each example $e \in E^*$ is of the form $e = \langle e_{\text{id}}, \langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle, e_{\text{ctx}}, e_{\text{raw}} \rangle$. e_{raw} is a set containing the raw data that corresponds to the neural inputs for the example.

The examples of NeuralFastLAS extend the CDPI of FastLAS described in Definition 2.2.2, but the probabilistic examples do not have penalties.

Example 4.0.2. Consider the neurosymbolic task defined in Example 3.0.2, then the NeuralFastLAS task is defined as $T = \langle B, M, E^*, \mathcal{Z} \rangle$ where

$B =$
```
num(0..27).
nn_label(1).  nn_label(2).  nn_label(3).
add(A, B, N)       :- N = NA + NB, nn(A, NA), nn(B, NB), num(N).
add_acc(A, NB, N)  :- N = NA + NB, nn(A, NA), num(NB), num(N).
mult(A, B, N)      :- N = NA * NB, nn(A, NA), nn(B, NB), num(N).
mult_acc(A, NB, N) :- N = NA * NB, nn(A, NA), num(NB), num(N).
```

$M_h =$
```
f(var(nn_label), var(nn_label), var(nn_label), var(num)).
```

$M_b =$
```
add(var(nn_label), var(nn_label), var(num)).
add_acc(var(nn_label), var(num), var(num)).
```

The predicate `nn/2` is a reserved predicate in NeuralFastLAS where the first argument is the identifier label for that input and the second argument represents the latent label of the corresponding raw data (the values of the identifiers are given by `nn_label/1`). For example, consider the datapoint $d = \langle (, ,), 14 \rangle$, then an answer set containing the correct neural network predicions will contain the literals `nn(1, 5)`, `nn(2, 0)` and `nn(3, 9)`.

The predicate `add/3` corresponds to directly adding two of the neural network outputs, and the predicate `add_acc/3` corresponds to adding a network output to an accumulated result from another operation. Similar is true for `mult/3` and `mult_acc/3`.

An example of a probabilistic example for this task would be

$$e_d = \langle e_{\text{id}}, \langle \{14\}, \{n \in \{0, 1, ..., 27\} : n \neq 14\} \rangle, \varnothing, \langle 5, 0, 9 \rangle \rangle$$

4.1 The NeuralFastLAS Pipeline

To begin, we give an overview of the algorithm and state its differences from the original FastLAS algorithm described in section 2.3:

1. **Abduction:** This step differs from the original FastLAS algorithm in that it must find all valid combinations of possible neural network outputs that explain a given label. In OPL tasks, each network output choice corresponds to a unique possibility. In Non-OPL tasks, possibilities are grouped by the network choices they correspond to.

2. **Compute SAT-sufficient Subset:** There are no changes that are strictly necessary for the NeuralFastLAS algorithm, but we introduce many new mode bias constraints to restrict the search space and remove redundant rules which leads to vast performance improvements. These constraints are task specific, some example usages are given in section 4.3.

3. **Generalisation:** The generalisation step remains consistent with the original FastLAS algorithm.

4. **Optimisation:** The optimization of rules must be more general to account for the probabilistic nature of the possibilities, we also use a unique pruning criteria to prune the opt-sufficient subset S_M^{opt}. It is necessary to perform different optimizations so that we can still guarantee that S_M^{opt} will contain the optimal solution.

5. **Train Neural Network:** With the opt-sufficient subset, we can train the neural network and additionally learn a probability distribution on S_M^{opt} in an end-to-end manner. This step is completely novel and chapter 5 is dedicated to this step.

6. **Solving:** The solving stage again uses a different concept of optimal solution than the original FastLAS algorithm since it must take into account the probabilities of labels predicted by the neural network trained in Step 5.

While the solution found is dependant on the convergence of the neural network, we prove a sufficient, but not necessary condition for the correctness of the NeuralFastLAS algorithm in Theorem 4.5.6. Furthermore, we show that the NeuralFastLAS can still return the correct solution when the neural network is poorly trained in Example 4.5.7.

4.2 Abduction

The set of choices of possible network outputs $\mathcal{Z}$ is converted into a program $P_{\mathcal{Z}}$. $\mathcal{Z}$ will be of the form $\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times ... \times \mathcal{Z}_n$ where the pipeline takes n raw data inputs per datapoint. For each $\mathcal{Z}_i$, we add the choice rule

```
1 { nn(i, Z_{i,1}), nn(i, Z_{i,2}), ..., nn(i, Z_{i,m}) } 1.
```

to $P_{\mathcal{Z}}$, which states that we can choose one of the possible latent labels for each input. Using the $abduce_possibilities(T, e)$ algorithm from FastNonOPL [12], we can define the neural abduction algorithm described in Algorithm 1. The symbol p_{as} represents the answer set of the bottom program that the possibility p corresponds to.

Algorithm 1 abduce_neural_possibilities(T, e)

$C = \bigcup_{A \in \text{AS}(P_{\mathcal{Z}})} A$
$pp_z = \varnothing$ for $z \in \mathcal{Z}$
Create a WCDPI $e' = \langle e_{\text{id}}, \infty, \langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle, e_{\text{ctx}} \cup P_{\mathcal{Z}} \rangle$
poss $= abduce_possibilities(T, e')$
for $p \in$ poss **do**
  $z' = p_{\text{as}} \cap C$
  $pp_{z'} = pp_{z'} \cup \{p\}$
end for
return $\{pp_z : z \in \mathcal{Z}\}$

Intuitively, the algorithm computes all possibilities of an probabilistic example by mapping it into a WCDPI and using $abduce_possibilities(T, e)$ [12] to get all possibilities of the WCDPI. The algorithm leverages the fact that $abduce_possibilities(T, e)$ begins by computing all of the answer sets $AS = \text{AS}(\text{bottom}(B \cup e_{\text{ctx}} \cup P_{\mathcal{Z}}))$ and then finding possible fixes for each $A \in AS$, as described in subsection 2.3.1. Assuming that $B \cup e_{\text{ctx}}$ contains no choice rules, then each answer set in AS maps uniquely to a possible latent label that satisfy the constraints in B (if any). If the task is Non-OPL, the $abduce_possibilities(T, e)$ will generate a set of possible fixes for each of the answer sets.

Once the possibilities have been generated by *abduce_possibilities*, we iterate through the answer sets to group them by their network latent choices. C represents the flattened set of all possible `nn(`i`,`$\mathcal{Z}_{i,j}$`)` which we use to extract the latent choices z' to deduce which group of possibilities a possibility should belong to. If the task is an OPL task, then each possibility will correspond to a unique neural latent output.

At the end of this stage, each example will have a set of groups of possibilities for each valid neural latent output.

4.3 Computing the SAT-Sufficient Subset

The first stage of NeuralFastLAS can produce a huge number of possibilities. Consider Example 3.0.2, each example will produce $10^3 = 1000$ possibilities. With many examples, the number of possibilities can make the task very computational intensive. A possible bottleneck in the NeuralFastLAS algorithm is computing the SAT sufficiency subsets since this is computed for each possibility individually.

To reduce the bottleneck, we may want to find ways to prune S_M by expressing more constraints on the mode bias. By providing more information to NeuralFastLAS about the nature of each predicate in the mode bias, we can restrict the search space so that Clingo produces fewer answer sets when computing $C^+(T, p)$ and $C^-(T, p)$, hence improving the performance of the algorithm while maintaining a notion of optimality.

In this section, we look at some of the mode bias constraints added to NeuralFastLAS.

4.3.1 Symmetric Arguments

Consider a situation where the SAT-sufficiency stage of a task generates the following rules:

```
f(V1,V2,V3) :- mult(V1,V2,V4), add(V1,V4,V3).
f(V1,V2,V3) :- mult(V1,V2,V4), add(V4,V1,V3).
f(V1,V2,V3) :- mult(V2,V1,V4), add(V1,V4,V3).
f(V1,V2,V3) :- mult(V2,V1,V4), add(V4,V1,V3).
```

We know that multiplication and addition are symmetric operations, so we can see that all four of these rules are equivalent.

Definition 4.3.1 (Symmetric Arguments)**.** Let P be an ASP program. Let p be a predicate with N arguments and let $S \subseteq \{1, 2, ..., N\}$ be a set of indices of the arguments of p, we say that p *is symmetric in arguments* S if for any $A \in \text{AS}(P)$, $p(V_1, V_2, ..., V_N) \in A$, then any permutation of the symmetric arguments (but keeping the non-symmetric arguments the same) is also in A.

Example 4.3.2. Let P be defined as

$$P = \left(\begin{array}{l} \texttt{num(0..100)} \\ \texttt{add(A, B, C) :- C = A + B, num(A), num(B).)} \end{array} \right)$$

Then P only has one answer set A, and for any `add(a, b, c)` $\in A$, we can also see that `add(b, a, c)` $\in A$, so `add/3` is symmetric in its first and second arguments.

To encode symmetric arguments in ASP during the SAT sufficiency stage of NeuralFastLAS, for every literal `l` in $\langle M_h, M_b \rangle$ with `N` arguments we create the predicate `sym_aux/4` to record the variable at each index of `l`. The first argument is the predicate name, the second argument is the total number of arguments, and the last two arguments represent an index and the variable at that index. For each index $0 <$ `M` $<$ `N`, we produce the rule

```
sym(l, N, M, ARGM) :- in(l(ARG1, ARG2, ..., ARGN)).
```

where the `in/1` predicate represents a literal that is in the body of the generated rule, as in the original FastLAS algorithm described in subsection 2.3.2. Furthermore, if `l` is symmetric in indices S, then for each `i` $\in S$, we add the atom

```
symmetric(l, N, i).
```

Finally, we can enforce symmetric by enforcing that if the predicate `l` occurs in the rule, the symmetric arguments must be ordered in terms of their index, this prevents many answer sets with the different permutations of the symmetric arguments being generated.

```
:- symmetric(L, N, I), symmetric(L, N, J),
   sym(L, N, I, ARG_I), sym(L, N, J, ARG_J),
   I < J,
   var_smaller(ARG2, ARG1).
```

which can be read that if the indices `I` and `J` of `L/N` are symmetric, then if `ARG_I` is at position `I` and `ARG_J` is at position `J` with `I` $<$ `J`, then it cannot be the case that `ARG_I` is smaller as a variable than `ARG_J`.

The predicate `var_smaller/2` just defines a lexicographical ordering of the variables.

4.3.2 Input and Output Variables

Another very significant way to constrain the search space is the try to capture the semantic intentions of each of the arguments in a predicate. As a demonstration, consider the following example:

Example 4.3.3. Let R denote the following rule

```
f(V1,V2,V3) :- mult(V4,V5,V5), add(V4,V3,V1), mult(V4,V3,V4).
```

The `f(V1, V2, V3)` head is intended to encode a mathematic function, so the intention is that we give arguments `V1` and `V2` to be used in the calculation, and `V3` is the result of the calculation.

However, in this rule, we can see that the variable `V3` is not used as we intended it to be used. We would expect `V3` to be the final argument of one of the body predicates (which correspond to the output of that predicate in the case of `add/3` and `mult/3`).

In order to formalise the problem in the above example, we introduce the concept of input and output variables:

Definition 4.3.4 (Input and Output Arguments)**.** We extend $M = \langle M_h, M_b \rangle$ so that each predicate bias also contains information about which arguments are inputs and outputs.

Let r be a rule h :- $b_1, b_2, ..., b_n$. Then we introduce the following two notions:

1. We say that h *is compatible with the input and output mode declarations* if :
   (a) For any variable V which appears in an input argument of h, there exists a body literal b_i such that V appears in an output argument of that literal.
   (b) For any variable V which appears in an input argument of a body literal b_i, either V appears in an output argument of the head or there exists a body literal b_j such that $j < i$ and V appears in an output argument of b_j.
   (c) Each output variable is unique, meaning that a variable V can appear in exactly one output argument. More formally, if V_i and V_j are both output variables, then $V_i \neq V_j$.
2. Furthermore, we can introduce an additional, optional constraint to say that r *consumes all of its outputs* if:
   (a) Every variable V which appears in an output argument of h must appear in the input argument of a body literal.
   (b) For every variable V which appears in an output argument of a body literal b_i, V must appear in the input argument of a body literal b_j such that $j > i$.

We use `var(-TYPE)` to represent output arguments in the mode bias and `var(+TYPE)` to represent input arguments (see Example 4.3.6 for an example mode bias).

Note that some arguments can be neither inputs nor outputs.

Remark 4.3.5. Note that the definition of input and output arguments is different to previous literature [16, 17], this is because the notion of input and output arguments is generally intended to capture evaluation semantics in Prolog. Definition 4.3.4 instead

introduces a new notion of input and output arguments for ASP that captures the semantic meaning intended with the predicates. A key difference between the definition in work and [16] is that the input and output arguments of the head predicate are treated differently - specifically, in this work we allow the input arguments of the head to appear in output arguments of the body. This is important for rules where the head predicate represents the result of some computation, for example.

Example 4.3.6. Consider the rule from Example 4.3.3

```
f(V1,V2,V3) :- mult(V4,V5,V5), add(V4,V3,V1), mult(V4,V3,V4).
```

together with mode biases

$$M_h = \big(\texttt{f(var(-num), var(-num), var(+num))} \big)$$
$$M_b = \left(\begin{array}{l} \texttt{add (var(+num), var(+num), var(-num))} \\ \texttt{mult(var(+num), var(+num), var(-num))} \end{array} \right)$$

The rule above does not respect the input and output mode declarations since the final argument `V3` of the head literal is an input argument but `V3` does not appear in the output argument of any of the body literals.

Consider now the rule

```
f(V1,V2,V3) :- mult(V1,V2,V4), add(V4,V2,V3), mult(V1,V4,V5).
```

This rule does respect the input and output mode declarations, but it does not consume all of its outputs since `V5` appears in the output of the final body literal but does not appear in any input arguments. Removing the last literal would create a rule that respects the input and output mode biases and also consumes all of its outputs.

To encode input and output constraints in FastLAS, part of the constraint is enforce in ASP, and the rest is in C++. We choose to ignore the ordering constraints in ASP since checking all the ordering in ASP will mean we have to consider every possible permutation of body literals for each rule, which will increase the number of answer sets by a combinatorial factor. Instead we enforce Definition 4.3.4 without the ordering requirement in ASP and then checking the ordering in C++ for the sake of performance.

The ASP encoding is very simple to encode, we define the new predicates `input_arg/2` and `output_arg/2` where the first argument stores the literal that the argument was derived from, and the second argument denotes the actual variable. Let `f/N` be a body literal with `N` arguments, suppose that `M` is an output argument, we add the rule

```
output_arg(l(ARG1, ARG2, ..., ARGN), ARGM) :- in(l(ARG1, ARG2, ..., ARGN)).
```

to the SAT-sufficiency stage as described in subsection 2.3.2. If, on the other hand, `M` is an input argument, we add the rule

```
input_arg(l(ARG1, ARG2, ..., ARGN), ARGM) :- in(l(ARG1, ARG2, ..., ARGN)).
```

to the SAT-sufficiency stage.

Next, to enforce Properties 1(a) and 1(b) of Definition 4.3.4 (without the ordering property as discussed earlier), we add the constraint

```
:- input_arg(L1, ARG), not output_arg(L2, ARG), in(L2), L1 != L2.
```

which says that an argument cannot be an input to some literal without being an output from some other literal.

Property 1(c) of Definition 4.3.4 is enforced with the constraint:

```
:- output_arg(L1, ARG), output_arg(L2, ARG), L1 != L2.
```

to ensure that there does not exist two literals with the same output variable.

Finally, should we wish to also enforce the constraint that every output argument is consumed as in Property 2 of Definition 4.3.4 , we add the constraint

```
:- output_arg(L1, ARG), not input_arg(L2, ARG), in(L2), L1 != L2.
```

4.4 Optimisation

At this stage, we now have the generalisation $G(T)$ of the rules from $C^+(T)$. We proceed by defining the following notion of neural optimality, which is slightly different from the normal FastLAS notion of optimality since probabilistic examples do not have penalties, instead we favour generalisation only with respect to coverage and length of the rules.

Definition 4.4.1 (Neural Optimisations)**.** Given a rule r, we define the *optimisations of* r, denoted $\text{neural_opt}(r)$, to be the set of rules r' that satisfy the following constraints:

1. r' is a subrule of r
2. There does not exist a rule r'' such that $r'' < r'$ and r' and r'' cover the exact same possibilities.
3. There does not exist any example $e \in E^*$ such that for every $p \in \text{possibilities}(e)$, r' is a subrule of a rule in $C^-(T, p)$.
4. r' is a smallest such rule that satisfies (1), (2), (3) and (4).

The opt-sufficient subset is then defined as S_M^{opt} where

$$S_M^{\text{opt}} = \bigcup_{r \in G(T)} \text{neural_opt}(r)$$

Item 3 is a pruning criteria introduced in NeuralFastLAS. Intuitively, item 3 says that if a rule violates every possibility of an example, then it is guaranteed to not be included in the final solution so we can prune it at the optimisation stage.

4.4.1 Probabilistic Opt-Sufficiency

To make guarantees on the optimality of NeuralFastLAS, we have to define a new notion of optimality to account for the uncertainty in the labels.

Definition 4.4.2 (The Collapse of a NeuralFastLAS Task)**.** Let $T = \langle B, M, E^*, \mathcal{Z} \rangle$ be a NeuralFastLAS task, we define the *collapse of* T into a normal FastLAS task $T' = \langle B, M, E \rangle$ by performing the following transformation:

For each example $e^* = \langle e_{\text{id}}, \langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle, e_{\text{ctx}}, e_{\text{raw}} \rangle \in E^*$, let z_{e^*} be the corresponding ground truth latent concepts for the elements of e_{raw}, then we can construct the WCDPI $e = \langle e_{\text{id}}, \infty, \langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle, e_{\text{ctx}} \cup z_{e^*} \rangle$ and add this to the set E in T'.

Intuitively, T' represent the learning task that uses the ground truth latent concepts of the raw data.

Definition 4.4.3 (Probabilistic Opt-Sufficiency)**.** We say that a set O is *probabilistically opt sufficient for* T if O contains an optimal solution of T' or T' is unsatisfiable.

We will now process to prove that the NeuralFastLAS algorithm guarantees probabilistic opt-sufficiency. We break the proof into lemmas for ease of reading.

In the following proofs, we will use the following notation. Let $T' = \langle B, M, E^+ \rangle$ be the collapse of the NeuralFastLAS task $T = \langle B, M, E^*, \mathcal{Z} \rangle$. For any example $e^* \in E^*$, there is a bijective mapping to an example $e \in E^+$. Let $z_{e^*} \in \mathcal{Z}$ denote the ground truth latent labels for the example e^* and pp_z represents the group of possibilities of e^* that take z as their network choices as in Algorithm 1. By construction, we have that $pp_{z_{e^*}} = \text{possibilities}(e)$, for every canonical pair (e, e^*).

Lemma 4.4.4. Let T be a NeuralFastLAS task and T' be the collapse of T into a normal FastLAS task, then $C^+(T') \subseteq C^+(T)$

Proof. Let $r \in C^+(T')$, then $r \in C^+(T', \tilde{p})$ for some possibility $\tilde{p}$. Let $\tilde{e}$ denote the example that $\tilde{p}$ belongs to in T' and $\tilde{e}^*$ represent the corresponding example in T, then it follows from $pp_{z_{\tilde{e}^*}} = \text{possibilities}(\tilde{e})$ that $\tilde{p} \in pp_{\tilde{e}^*}$. Finally, we can deduce that $r \in C^+(T', \tilde{p}) = C^+(T, \tilde{p}) \subseteq C^+(T)$, so $r \in C^+(T)$. □

Lemma 4.4.5. Suppose that $C^+(T') \subseteq C^+(T)$, then $G(T') \subseteq G(T)$.

Proof. Assume that $C^+(T') \subseteq C^+(T)$, now choose a rule $r_G \in G(T')$. Then r_G has the following properties in T' by Definition 2.3.13:

1. $c_{T',r_G} \neq \varnothing$ - we add the extra subscript argument to distinguish which task the characteristic ruleset belongs to.
2. There is no rule $r' \in S_M$ such that r is a strict subrule of r' and $c_r = c_{r'}$

Since the generalisation step is the same in NeuralFastLAS as in normal FastLAS, it suffices to show that r_G has the same properties in T:

1. Since $c_{T',r_G} \neq \varnothing$, this means there is at least one rule $r \in c_{T',r_G}$, such that $r \in C^+(T')$ and r is a subrule of r_G. Since $C^+(T') \subseteq C^+(T)$, then $r \in C^+(T)$, so $r \in c_{T,r_G}$ and $c_{T,r_G} \neq \varnothing$.
2. Since both tasks have the same S_M by definition of the collapse of T, it follows that there is still no rule $r' \in S_M$ such that r is a strict subrule of r' and $c_r = c_{r'}$.

Hence, we have shown that $r_G \in G(T)$. □

Lemma 4.4.6. Let O be the opt-sufficient subset of T' produced by the normal FastLAS algorithm. If $G(T') \subseteq G(T)$, then for any optimal solution H of T' contained in O, $H \subseteq S_M^{\text{opt}}$.

Proof. Choose some $r \in H$, where H is any optimal solution of T' contained in O. The following is true by definition of optimisations in normal FastLAS, given in Definition 2.3.14

1. r is a subrule of some $r_G \in G(T')$
2. There is no rule r' such that $r' < r$ and r and r' cover the exact same possibilities.

Now we wish to show that $r \in S_M^{\text{opt}}$, so we must prove that r is a neural optimisation, so satisfies the properties given in Definition 4.4.1

1. r is a subrule of r_G, since $G(T') \subseteq G(T)$, we have $r_G \in G(T)$.
2. Suppose for a contradiction that there existed a rule r' such $r' < r$ and r and r' cover the exact same possibilities. Let $\text{pos_cov}(T, r)$ be the set of possibilities in T that r covers, then $\text{pos_cov}(T', r)$ will be equal to the restriction of $\text{pos_cov}(T, r)$ into $\text{possibilities}(T)$:

   $$\text{pos_cov}(T', r) = \text{pos_cov}(T, r) \cap \text{possibilities}(T')$$

   We have $\text{pos_cov}(T, r) = \text{pos_cov}(T, r')$, so by the properties of sets, we get

   $$\text{pos_cov}(T, r) = \text{pos_cov}(T, r') \Rightarrow \text{pos_cov}(T', r) = \text{pos_cov}(T', r')$$

   Hence r' is a shorter rule than r that covers the same possibilities as r in T', so r is not an optimisation of r_G in T'. Hence we have a contradiction, and there exists no such r'.
3. Suppose for a contradiction that there did exist such an example $\tilde{e}^* \in E^*$, so $\forall p \in \text{possibilities}(\tilde{e}^*)$, r is a subrule of some rule in $C^-(T, p)$. By definition, we have $pp_{z_{\tilde{e}^*}} \subseteq \text{possibilities}(\tilde{e}^*)$ so r does not cover any possibilities in $pp_{z_{\tilde{e}^*}}$.

   Let $\tilde{e}$ be the corresponding example in T', since r is part of an optimal solution, then r must cover at least one possibility for every $e \in E^+$. In particular, r must cover a possibility in $\text{possibilities}(\tilde{e}) = pp_{z_{\tilde{e}^*}}$ so r must cover a possibility in $pp_{z_{\tilde{e}^*}}$ hence we reach a contradiction.

Finally, we have shown that r satisfies the conditions to be in S_M^{opt}. □

Theorem 4.4.7. S_M^{opt} is probabilistically opt-sufficient.

Proof. Suppose that T' is satisfiable, then by Lemma 4.4.4, Lemma 4.4.5 and Lemma 4.4.6, we deduce that for any optimal solution H of T' contained in O, $H \subseteq S_M^{\text{opt}}$. Since T' is satisfiable, it contains at least one optimal solution H^* and by the above, we must have $H^* \subseteq S_M^{\text{opt}}$, so S_M^{opt} is probabilistically opt-sufficient. □

4.5 Solving Stage

Finally, we define the NeuralFastLAS solving stage. This stage occurs after training the neural network, which is discussed in chapter 5.

The solving stage attempts to maximize the following optimization problem, motivated by [18]:

$$\begin{aligned} H^* &= \arg\max_H \prod_{\langle x,y\rangle \in \mathcal{D}} \sum_{z\in\mathcal{Z}} P(y, z|B, x, H, \theta) \\ &= \arg\max_H \prod_{\langle x,y\rangle \in \mathcal{D}} \sum_{z\in\mathcal{Z}} P(y|B, H, z) P_{\sigma^*}(H|B) P_\theta(z|x) \end{aligned}$$

where

$$P(y|B, H, z) = \begin{cases} 1 & \text{if } B \cup H \cup z \models y \\ 0 & \text{otherwise} \end{cases}$$

To make the optimization problem possible to solve in FastLAS, we perform the following sampling techniques:

1. Take the product only over the examples used in the rest of the pipeline rather than the entire dataset. If we assume the sample is drawn i.i.d. from the original dataset, then this is a correct sampling technique.

2. Instead of taking the sum over $z \in \mathcal{Z}$, we only consider the maximum value. Intuitively, to see that this is a fair sampling technique, recall that the solving stage occurs after training the neural network, so at this point we can assume the network makes good predications. This means that, on average, $P_\theta(z|x)$ will be larger for the correct z, and small for all of the incorrect z, hence the maximum will be a good representation of the actual value.

With these sample techniques, we can now define the optimisation function used in NeuralFastLAS:

Definition 4.5.1 (NeuralFastLAS Optimisation Function)**.** Let T be a NeuralFastLAS task and θ represent the parameters of the trained neural network. A hypothesis H is said to be optimal if it satisfies

$$H = \arg\max_H \prod_{e\in E^*} \max_{z\in\mathcal{Z}} P(e_{\text{pi}}^{\text{inc}}|B, H, z) P_{\sigma^*}(H|B, \theta) P_\theta(z|e_{\text{raw}}) \tag{4.1}$$

The function in Equation 4.1 is very far from the form of a normal FastLAS function: $S^{\text{pen}} + S^{\text{rule}}$. To translate the problem into a scoring function closer to that of a normal scoring function, we can perform some transformations. First, we define:

$$\text{cov}(H, e) = \{z \in \mathcal{Z} : \exists p \in pp_z \text{ such that } B \cup H \cup p_{\text{ctx}} \models e_{\text{pi}}^{\text{inc}}\}$$

then we perform the following steps to transform the optimisation problem:

$$\arg\max_H \prod_{e\in E^*} \max_{z\in\mathcal{Z}} P(e^{\text{inc}}_{\text{pi}}|B,H,z)P_\sigma(H|B)P_\theta(z|e_{\text{raw}})$$

$$= \arg\max_H \prod_{e\in E^*} \max_{z\in\text{cov}(H,e)} P_{\sigma^*}(H|B)P_\theta(z|e_{\text{raw}})$$

$$= \arg\min_H -\log\left[\prod_{e\in E^*} \max_{z\in\text{cov}(H,e)} P_\sigma(H|B)P_\theta(z|e_{\text{raw}})\right]$$

$$= \arg\min_H \sum_{e\in E^*} -\log\left[\max_{z\in\text{cov}(H,e)} P_\sigma(H|B)P_\theta(z|e_{\text{raw}})\right]$$

$$= \arg\min_H \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} [-\log P_\sigma(H|B) - \log P_\theta(z|e_{\text{raw}})]$$

$$= \arg\min_H \sum_{e\in E^*} -\log P_\sigma(H|B) + \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})$$

$$= \arg\min_H \left[-|E^*|\log P_\sigma(H|B) + \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})\right] \tag{4.2}$$

The probability $P_\sigma(H|B)$ is a prior distribution on H. Previous works [19] have used the probability distribution

$$P'_\sigma(H|B) = \frac{6}{\pi^2|H|^2}$$

However, we introduce a new prior

$$P_\sigma(H|B) = (e-1)e^{-|H|} \tag{4.3}$$

that is more compatible with the FastLAS transformations we wish to achieve. By substituting this into Equation 4.2, we can continue to simplify

$$\arg\min_H \left[-|E^*|\log P_\sigma(H|B) + \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})\right]$$

$$= \arg\min_H \left[-|E^*|\log\left[(e-1)e^{-|H|}\right] + \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})\right]$$

$$= \arg\min_H \left[-|E^*|\log(e-1) + |E^*||H| + \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})\right]$$

$$= \arg\min_H \left[|E^*||H| + \sum_{e\in E^*} \min_{z\in\text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})\right] \tag{4.4}$$

Now we have two distinct parts of our optimisation: the first is a value that scores the hypothesis, and the second is a value that scores the best set of network choices z such that one of the possibilities that assume z as the network parameters is covered by H. This is a lot closer to the original FastLAS function which is of the form

$$S(H,T) = S'(H,T) + \sum_{e \in \text{Uncov}(H,T)} e_{\text{pen}}$$

where S' is decomposable (see Definition 2.3.5). The NeuralFastLAS scoring function differs in that we are not concerned about which examples are not covered, but instead we are concerned about covering the possibilities that correspond to the best latent choices.

The probability $-\log P_\theta(z|e_{\text{raw}})$ can be broken down further in the case that e_{raw} represents multiple inputs: Let $e_{\text{raw}} = \langle x_1, x_2, ..., x_n \rangle$, then we have

$$-\log P_\theta(z|e_{\text{raw}}) = -\log \prod_{i=1}^{n} P_\theta(z_i|x_i) = \sum_{i=1}^{n} -\log P_\theta(z_i|x_i)$$

With this information, we can construct the ASP program P_{solve} that will solve our final task and return the optimal hypothesis, this is based roughly on the solving stage of the original FastLAS [13].

P_{solve} consists of the following elements:

1. For each hypothesis $h \in S_M^{\text{opt}}$, we add to P_{solve} the choice rule and weak constraint:

   0 { in_h(h_{id}) } 1.
   :~ in_h(h_{id}).[$|E^*||H|$@1, in_h(h_{id})]

2. For each example $e \in E^*$, we have the constraint

   :- not covered(e_{id}).

   Furthermore, we say that an example is covered if at least one of its possibility groups is covered and a possibility group is covered if at least one of its possibilities is covered.

   Each possibility group corresponds to a latent labelling $z \in \mathcal{Z}$. In ASP, we will identify z by $\mathtt{z}_{\text{id}}$. For each possibility group pp_z, we perform the following:

   (a) Add the rule

   covered(e_{id}) :- poss_group_covered(e_{id},z_{id}).

   (b) For each possibility $\mathtt{p} \in pp_z$, we add the rule

   poss_group_covered(e_{id},z_{id}) :- not poss_not_covered(p_{id}).

   (c) Add the atom

   poss_group_penalty(e_{id},z_{id},$P_\theta(z|x)$).

   (d) For each $\mathtt{p} \in pp_z$, to define poss_not_covered, we use the same logic as in subsection 2.3.5: Define for convenience

   $$O^+(T,a,\mathtt{p}) = \{h : h \in S_M^{\text{opt}},\ \exists r \in C^+(T,a,\mathtt{p}) \text{ such that } h \leq r\}$$
   $$O^-(T,a,\mathtt{p}) = \{h : h \in S_M^{\text{opt}},\ \exists r \in C^-(T,a,\mathtt{p}) \text{ such that } h \leq r\}$$

   Then we create the rules

```
poss_not_covered(p_id) :- ⋀_{h∈O⁺(T,a,p)} not in_h(h_id).
```

and for each $\mathtt{a} \in \mathtt{p}^{\mathrm{exc}}_{\mathrm{pi}}$ and $\mathtt{h} \in O^{-}(T, \mathtt{a}, \mathtt{p})$, we add the rule

```
poss_not_covered(p_id) :- in_h(h_id).
```

3. Finally, we add the rule

```
min_ex_penalty(Ex, Pen)
        :- example(Ex),
           Pen=#min{ P : poss_group_covered(Ex, Z),
                         poss_group_penalty(Ex, Z, P) }.
```

and the weak constraint

```
:~ min_ex_penalty(Ex, P).[P@1, Ex].
```

With this weak constraint combined with the weak constraint defined in Step 1, Clingo will attempt to minimize over the sum of the hypothesis size penalty and the minimum example penalties which is consistent with Equation 4.4.

We solve this using Clingo to gain as answer set from which we can build an optimal hypothesis H^*. We denote this solution by $H^* = \text{neural_solve}(T, \theta)$

Remark 4.5.2. One small technical note to bare in mind is that Clingo only supports integer penalties. Hence, when a probability is encoded in Clingo, we encoded the value in Clingo using the map $v \mapsto \text{round}(100 \times v)$. The multiplication value is determined as a hyperparameter depending how many decimal places is considered important. For most examples, scaling by 100 is sufficient. The hypothesis size penalty is similarly scaled.

4.5.1 Optimality of the Solving Stage

We would like to guarantee that NeuralFastLAS returns the correct solution of a task. However, it is dependant on the training of the neural network. We begin by formalising the definition of an correct NeuralFastLAS solution, which says an solution is correct if it is also a solution to the normal FastLAS task that uses the ground truth labels.

Definition 4.5.3 (NeuralFastLAS Correctness)**.** We say that a solution H^* to a Neural-FastLAS task T is *correct* if H^* is also an optimal solution to the collapse of T, T'.

Remark 4.5.4. Note that a correct solution has a different meaning to an optimal solution. A solution of a NeuralFastLAS task is optimal if it minimizes the scoring function, whereas a correct solution of a NeuralFastLAS task is also an optimal solution of the normal FastLAS task that uses the ground truth labels.

We demonstrate that if the network trains "almost perfectly", then NeuralFastLAS returns a correct solution. It is very important to emphasise that the condition of the network training "almost perfectly" is a **sufficient, but not necessary condition**, the neural network can be trained poorly but NeuralFastLAS may still find the optimal solution.

Definition 4.5.5 (Almost Perfectly Trained Networks)**.** A neural network with parameters θ is said to have trained almost perfectly for a task $T = \langle B, M, E^*, \mathcal{Z} \rangle$ if the following bound

is satisfied: Let H^* be a correct solution for T, then for any input x with ground truth latent label z^{gt}, we have

$$P_\theta(z^{\text{gt}}|x) \geq \frac{e^{|E^*||H^*|}}{1 + e^{|E^*||H^*|}} \tag{4.5}$$

In other words, the network predicts z^{gt} as the most likely label for x, but not necessarily with confidence (i.e. it is not the case that $p(z^{\text{gt}}|x) = 1$).

It follows that for any $z \neq z^{\text{gt}}$, we have $P_\theta(z|x) \leq 1 - P_\theta(z^{\text{gt}}|x)$, so by Equation 4.5, we get that

$$P_\theta(z|x) \leq \frac{1}{1 + e^{|E^*||H^*|}} \tag{4.6}$$

Now we attempt to prove that, in the case of almost perfectly trained networks, Neural-FastLAS always returns a correct solution.

Theorem 4.5.6. Let θ be the parameters of an almost perfectly trained network, then $H^* = \text{neural_solve}(T, \theta)$ is a correct solution of T.

Proof. Let H' be any optimal solution to the normal FastLAS task T', if we define HS_{cov} to be the set of hypotheses H such that for every example $e \in E^+$, H covers e then by definition of an optimal solution of a normal FastLAS task (where every example has infinite penalties) we must have

$$H' = \operatorname*{arg\,min}_{H \in HS_{\text{cov}}} |H|$$

Now let H^* be an optimal solution of T, we will attempt to show that H^* is correct, so H^* is equal to some optimal solution of T'. First, define

$$\text{score}(H) = |E^*||H| + \sum_{e \in E^*} \min_{z \in \text{cov}(H,e)} -\log P_\theta(z|e_{\text{raw}})$$

so then, by Equation 4.4, we have

$$H^* = \operatorname*{arg\,min}_{H} \text{score}(H)$$

First we show that the minimum over the sum of the examples is achieved when $H \in HS_{\text{cov}}$. For a given x with ground truth latent label z_x^{gt}, by the assumption of having an almost perfectly trained neural network, we have

$$\forall z \neq z_x^{\text{gt}}.\ P_\theta(z|x) < P_\theta(z_x^{\text{gt}}|x)$$

Hence, for a given H and example $e \in E^*$, if $z_{e_{\text{raw}}}^{\text{gt}} \in \text{cov}(H, e)$, then

$$\max_{z \in \text{cov}(H,e)} P_\theta(z|e_{\text{raw}}) = P_\theta(z_{e_{\text{raw}}}^{\text{gt}}|e_{\text{raw}})$$

But if $z_{e_{\text{raw}}}^{\text{gt}} \notin \text{cov}(H, e)$, then

$$\max_{z \in \text{cov}(H,e)} P_\theta(z|e_{\text{raw}}) < P_\theta(z_{e_{\text{raw}}}^{\text{gt}}|e_{\text{raw}})$$

Next, we will show that $H^* \in HS_{\text{cov}}$. To do this, suppose for a contradiction that that $H^* \notin HS_{\text{cov}}$, then H^* does not cover every example in E^+. Relating this to T, this means that there exists some example $e^* \in E^*$ such that $z_{e^*_{\text{raw}}}^{\text{gt}} \notin \text{cov}(H^*, e^*)$. Let H' be any optimal solution of T', we will show that $\text{score}(H^*) > \text{score}(H')$ to show that H^* is not

an optimal solution of T and reach a contradiction. To do this, we perform the following computation

$$\begin{aligned}
&\mathrm{score}(H^*) - \mathrm{score}(H') \\
&= |E^*|(|H^*| - |H'|) + \sum_{e \in E^*} \left[\min_{z \in \mathrm{cov}(H^*,e)} - \log P_\theta(z|e_{\mathrm{raw}}) - \min_{z \in \mathrm{cov}(H',e)} - \log P_\theta(z|e_{\mathrm{raw}}) \right] \\
&= |E^*|(|H^*| - |H'|) + \sum_{e \in E^*} \left[- \log \max_{z \in \mathrm{cov}(H^*,e)} P_\theta(z|e_{\mathrm{raw}}) + \log \max_{z \in \mathrm{cov}(H',e)} P_\theta(z|e_{\mathrm{raw}}) \right] \\
&= |E^*|(|H^*| - |H'|) + \sum_{e \in E^*} \left[- \log \max_{z \in \mathrm{cov}(H^*,e)} P_\theta(z|e_{\mathrm{raw}}) + \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) \right] \\
&= |E^*|(|H^*| - |H'|) + \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \notin \mathrm{cov}(H^*,e)}} \left[- \log \max_{z \in \mathrm{cov}(H^*,e)} P_\theta(z|e_{\mathrm{raw}}) + \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) \right] \qquad (*) \\
&\geq |E^*|(|H^*| - |H'|) + \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \notin \mathrm{cov}(H^*,e)}} \left[- \log \frac{1}{1 + e^{|H'||E^*|}} + \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) \right] \\
&\geq |E^*|(|H^*| - |H'|) + \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \notin \mathrm{cov}(H^*,e)}} \left[- \log \frac{1}{1 + e^{|H'||E^*|}} + \log \frac{e^{|H'||E^*|}}{1 + e^{|H'||E^*|}} \right] \\
&= |E^*|(|H^*| - |H'|) + \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \notin \mathrm{cov}(H^*,e)}} \log e^{|H'||E^*|} \\
&= |E^*|(|H^*| - |H'|) + \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \notin \mathrm{cov}(H^*,e)}} |H'||E^*| \\
&\geq |E^*|(|H^*| - |H'|) + |H'||E^*| \\
&= |E^*||H^*| > 0
\end{aligned}$$

Hence, we have shown that $\mathrm{score}(H^*)$ is not an optional solution for T, which contradicts our choice of H^*, hence it must be true that $H^* \in HS_{\mathrm{cov}}$.

The step $(*)$ is valid since the summand evaluates to 0 when $z_e^{\mathrm{gt}} \in \mathrm{cov}(H^*, e)$. To see this, we compute:

$$\begin{aligned}
&\sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \in \mathrm{cov}(H^*,e)}} \left[- \log \max_{z \in \mathrm{cov}(H^*,e)} P_\theta(z|e_{\mathrm{raw}}) + \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) \right] \\
= &\sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \in \mathrm{cov}(H^*,e)}} \left[- \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) + \log P_\theta(z_e^{\mathrm{gt}}|e_{\mathrm{raw}}) \right] = \sum_{\substack{e \in E^* \\ z_e^{\mathrm{gt}} \in \mathrm{cov}(H^*,e)}} 0 = 0
\end{aligned}$$

Finally, we will show that H^* is a minimum of HS_{cov} with respect to the size of the hypothesis. Suppose for a contradiction that this is not the case - so there is some $H' \in HS_{\text{cov}}$ such that $|H'| < |H^*|$ - then we will again show that this implies H^* is not an optimal solution of T.

Since $H', H^* \in HS_{\text{cov}}$, for any $e \in E^+$ we have $z_e^{\text{gt}} \in \text{cov}(H^*, e)$ and $z_e^{\text{gt}} \in \text{cov}(H', e)$, hence

$$\sum_{e \in E^*} \min_{z \in \text{cov}(H',e)} -\log P_\theta(z|e_{\text{raw}}) = \sum_{e \in E^*} -\log P_\theta(z_e^{\text{gt}}|e_{\text{raw}}) = \sum_{e \in E^*} \min_{z \in \text{cov}(H^*,e)} -\log P_\theta(z|e_{\text{raw}})$$

Now we compute

$$\begin{aligned}
\text{score}(H') &= |E^*||H'| + \sum_{e \in E^*} \min_{z \in \text{cov}(H',e)} -\log P_\theta(z|e_{\text{raw}}) \\
&= |E^*||H'| + \sum_{e \in E^*} \min_{z \in \text{cov}(H^*,e)} -\log P_\theta(z|e_{\text{raw}}) \\
&< |E^*||H^*| + \sum_{e \in E^*} \min_{z \in \text{cov}(H^*,e)} -\log P_\theta(z|e_{\text{raw}}) \\
&= \text{score}(H^*)
\end{aligned}$$

Hence we have again shown H^* is not optimal reaching a contradiction. Therefore it must be the case that there does not exist any $H' \in HS_{\text{cov}}$ such that $|H'| < |H^*|$. Finally, we have shown

$$H^* = \arg\min_{H \in HS_{\text{cov}}} |H|$$

So H^* is an optimal solution for T' as required, so H^* is a correct solution of the NeuralFastLAS task T. □

It is important to re-emphasis that Theorem 4.5.6 is a **sufficient, but not necessary condition**. NeuralFastLAS can find a correct solution even if the network does not train well. We demonstrate this in the following example:

Example 4.5.7. Consider the task of learning to multiply two numbers. In this example, we use a very simple task with one example and restrict the raw data inputs to represent numbers between 0 and 4 for the sake of demonstration. The inclusion for the example is $\{f(a, b, 3)\}$ and the correct rule is `f(A, B, N) :- mult(A, B, N).`

Suppose the network parameters θ return the probabilities given in the diagram below. It is clear that θ does not satisfy the notion of an almost perfectly trained network, however we will show that it still learns the correct rule. To see that the network does not satisfy the conditions of being almost perfectly trained, we can compute

$$P_\theta(z^{\text{gt}}|x) = P_\theta(3|x_1)P_\theta(1|x_2) = 0.3 \times 0.3 = 0.09$$

and

$$\frac{e^{|E^*||H^*|}}{1 + e^{|E^*||H^*|}} = \frac{e^2}{1 + e^2} \approx 0.88$$

to see that Equation 4.5 is not satisfied.

The network incorrectly predicts the first number as a 0 and the second number as a 4. However, for both inputs, the second most likely label corresponds to the correct label. Suppose that we have

$$S_M^{\text{opt}} = \begin{pmatrix} h_1 : \texttt{f(A, B, N) :- add(A, B, N).} \\ h_2 : \texttt{f(A, B, N) :- mult(A, B, N).} \end{pmatrix}$$

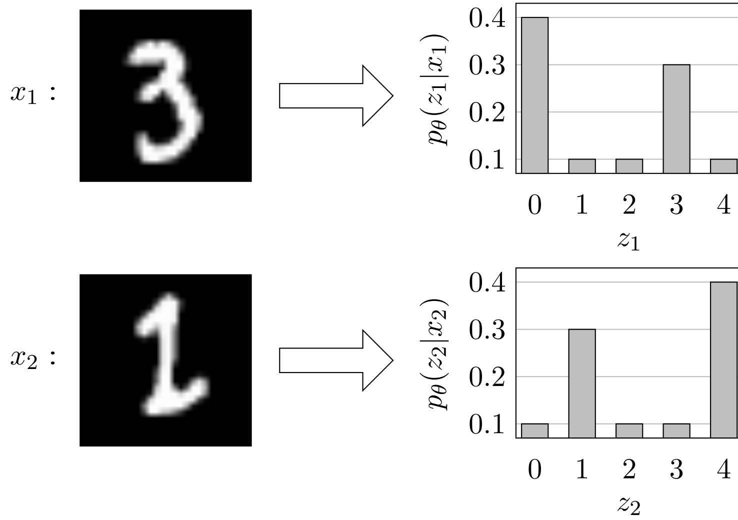

Figure 4.1: Example of Poorly Trained Neural Network

then the set of possible hypotheses is given by $\{h_1, h_2, h_3\}$ where $h_3 = h_1 \vee h_2$. We now compute score(H) for each H. To do this, we first find

$$\begin{aligned}
\text{cov}(h_1) &= \{(0,3),(1,2),(2,1),(3,0)\} \\
\text{cov}(h_2) &= \{(1,3),(3,1)\} \\
\text{cov}(h_3) &= \{(0,3),(1,2),(2,1),(3,0),(1,3),(3,1)\}
\end{aligned}$$

Note that the most likely label predicted by the neural network $\hat{z} = (0,4)$ does not appear in any of the above sets. The NeuralFastLAS algorithm is able to discard latent labels that are semantically nonsense: $0+4 \neq 3$ and $0 \times 4 \neq 3$, so it is not a valid latent label for this example.

Now we can compute

$$\begin{aligned}
\text{score}(h_1) &= |E^*||H| + \sum_{e \in E^*} \min_{z \in \text{cov}(h_1,e)} -\log P_\theta(z|x) \\
&= |E^*||h_1| - \log \max_{z \in \text{cov}(h_1,e)} P_\theta(z_1|x_1)P_\theta(z_2|x_2) \\
&= 1 \times 2 - \log(0.4 \times 0.1) \approx 5.21
\end{aligned}$$

$$\text{score}(h_2) = 1 \times 2 - \log(0.3 \times 0.3) \approx 4.41$$

$$\text{score}(h_3) = 1 \times 4 - \log(0.3 \times 0.3) \approx 6.41$$

Hence, we finally see that

$$h_2 = \arg\min_H \ \text{score}(H)$$

So h_2 is the optimal hypothesis. Hence, we have shown an example where NeuralFastLAS can solve the task correctly despite receiving inputs from a poorly trained network.

Chapter 5

End-To-End Training with NeuralFastLAS

The stages for the symbolic learning of the NeuralFastLAS architecture were described in the previous chapter. Now we turn our attention to understanding the neural stage of the architecture. Training NeuralFastLAS consists:

1. **Training the Perceptron Component:** The raw data is fed into a neural network which outputs probability vectors predicting the corresponding latent label for each raw data input. During the gradient descent step, NeuralFastLAS optimizes the parameters to improve the accuracy of the network.
2. **Learning a Posterior Distribution on the Rules:** NeuralFastLAS also uses an extra parameter which we denote θ_R such that softmax(θ_R) gives us a probability vector. Intuitively, we can see $p(r_i|\theta_R)$ as learnt probability score that rule r_i from S_M^{opt} covers a particular datapoint.

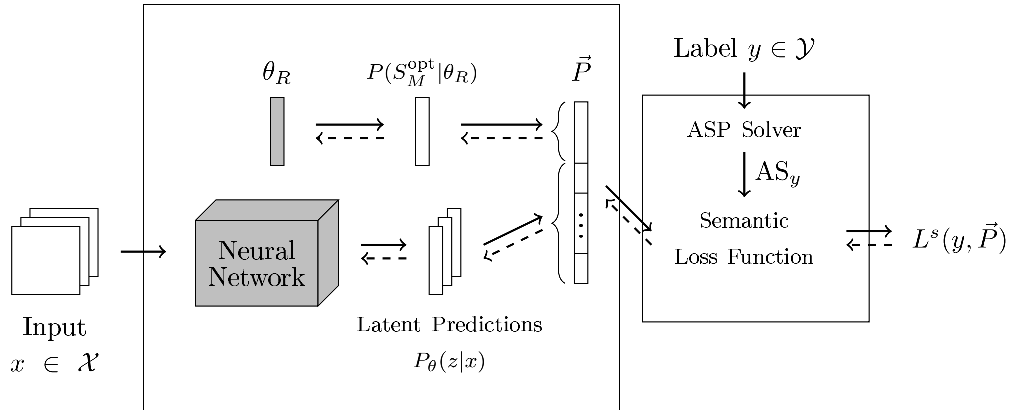

Figure 5.1: Schematic of the Neural Component of NeuralFastLAS. Gray sections show where gradient descent steps occur. The dashed line show where gradients are propagated. The two boxes denote the neural component (left) and semantic component (right)

The training of the neural component differs from previous end-to-end training methods [18, 20] in two ways: first, the use of the semantic loss function [5], and also in the novel use of a learnt posterior distribution of rules to guide learning.

In this chapter we explore how NeuralFastLAS uses the semantics loss to train the perceptron network.

5.1 Semantic Loss in NeuralFastLAS

The semantic loss function [5] is intended to capture how close a neural network's predictions are to satisfying some symbolic constraints. In the NeuralFastLAS architecture, we can use the semantic loss to capture which latent label predictions can be used to prove the downstream label and propogate this information back to the neural network during gradient descent.

To begin, we rewrite the semantic loss function in the notion of ASP (the original definition is discussed in section 7.2).

Definition 5.1.1 (Semantic Loss in ASP)**.** Let P be the program for which we are trying to compute the semantic loss, it will contain all of the background knowledge plus choice rules to represent all of the possible values of latent concepts that the network can predict. Let AS_y denote the set of answer sets of P that contain y, then with this formalisation, we can define the semantic loss as

$$L^s(y, p) \propto \log \sum_{A \in \text{AS}_y} \prod_{\substack{i \in \{0,\dots,n\} \\ X_i \in A}} p_i \prod_{\substack{i \in \{0,\dots,n\} \\ X_i \notin A}} (1 - p_i) \tag{5.1}$$

where p is a posterior probability vector outputted from the neural network and p_i denotes the predicted probability of atom X_i.

Previous symbolic reasoners [21] have used the semantic loss function with handwritten rules. NeuralFastLAS is different in that the network is trained before the final solution is found. This introduces a new problem: the many incorrect rules produce a lot of noise in the loss computation. To illustrate this, consider the following example:

Example 5.1.2. Recall the arithmetic task defined in Example 4.5.7 with downstream label $y = 3$, we will look at the answer sets produced by both of the rules in S_M^{opt}:

$$h_1 : \texttt{f(A, B, N) :- add(A, B, N).} \Rightarrow \left\{ \begin{array}{l} \text{AS}_1 := (\texttt{nn(a, 0)}, \texttt{nn(b, 3)}) \\ \text{AS}_2 := (\texttt{nn(a, 1)}, \texttt{nn(b, 2)}) \\ \text{AS}_3 := (\texttt{nn(a, 2)}, \texttt{nn(b, 1)}) \\ \text{AS}_4 := (\texttt{nn(a, 3)}, \texttt{nn(b, 0)}) \end{array} \right\}$$

$$h_2 : \texttt{f(A, B, N) :- mult(A, B, N).} \Rightarrow \left\{ \begin{array}{l} \text{AS}_5 := (\texttt{nn(a, 1)}, \texttt{nn(b, 3)}) \\ \text{AS}_6 := (\texttt{nn(a, 3)}, \texttt{nn(b, 1)}) \end{array} \right\}$$

The correct latent labels are $(\texttt{nn(a, 3)}, \texttt{nn(b, 1)})$ which corresponds to answer set AS_6. Only 1/6 of the possible answer sets contain the correct labels, whereas if just the ground truth rule h_2 were used then 1/2 of the answer sets would contain the correct label. Hence, incorrect rules decrease the strength of the signal from AS_6 propagated back to the network.

In order to try to compensate for the noisy signals produced by the incorrect rules, we can attempt to learn a probability distribution on the rules in S_M^{opt}. The hope is that the ground truth rules will generate more answer sets on average and the probabilities learnt will favour those ground truth rules. In turn, answer sets generated by the correct rule will be weighted higher and the effect of noise during gradient descent will decrease.

We use a variable vector θ_R that represents "scores" of the rules. Softmax is appplied to this vector to produce an probability distribution $p(r|\theta_R)$. It is preferable to interpret $\text{softmax}(\theta_R)$ as the probabilities rather than θ_R, since gradient update steps will likely produce an unnormalised vector - violating the constraints of a probability distribution.

Example 5.1.3. Continuing from Example 5.1.2, suppose that the network learns a posterior distribution on the rules that assigns a 10% probability to h_1 and a 90% probability to h_2. If we define

$$L_{\text{AS}_n} = \prod_{\substack{i \in \{0,\ldots,n\} \\ X_i \in \text{AS}_n}} p_i \prod_{\substack{i \in \{0,\ldots,n\} \\ X_i \notin \text{AS}_n}} (1 - p_i)$$

then we can write

$$L^s(\alpha, p) = -\log \sum_n L_{\text{AS}_n} = -\log \left(0.1 \sum_{i=1}^{4} L_{\text{AS}_i} + 0.9 \sum_{i=5}^{6} L_{\text{AS}_i} \right)$$

and so when we compute gradients and back-propagate them, AS_5 and AS_6 belonging to h_2 optimising the loss with respect to L_{AS_5} and L_{AS_6} will contribute more significantly due to their higher weighting and overshadow the answer sets generated by the incorrect rule.

In a way, we can consider the neural network to have two heads: the first is the perceptron component, the second is the rule probability predictor.

5.2 Computing the Semantic Loss

In this section, we explain the mechanism we use to compute the semantic loss with $p_\theta(z|x)$ and $p(r|\theta_R)$. Suppose we have an example $e^* = \langle e_{\text{id}}, \langle e_{\text{pi}}^{\text{inc}}, e_{\text{pi}}^{\text{exc}} \rangle, e_{\text{ctx}}, e_{\text{raw}} \rangle$ and neural network parameters θ and θ_R, we define the atoms X_i for the semantic loss by:

$$X_i = \begin{cases} \texttt{use(}i\texttt{)} & \text{for } i < |S_M^{\text{opt}}| \\ \texttt{nn(}j\texttt{, }k\texttt{)} & \text{otherwise, where } i = |S_M^{\text{opt}}| + jn + k \end{cases} \tag{5.2}$$

where n is the number of latent concepts for each raw data input. In essence, the first $|S_M^{\text{opt}}|$ elements mapped to by X_i represent possible choices for rules, and the rest of the elements represent the enumeration over the possible latent concept labels. Note that in the second case, j starts from 0 and k indexes the possible latent concept values.

To generate the answer sets to compute the semantic loss y, let P be the program defined in Definition 5.1.1. We add to P the following components to create P_y:

1. We use the constraint

   `:- not` y`.`

   to restrict the answer sets only to those that prove y.

2. For each rule $r_i \in S_M^{\text{opt}}$, we add the following rule to P:

   $\text{head}(r_i)$ `:- use(`i`),` $\text{body}(r_i)$`.`

3. Since NeuralFastLAS tasks are non-recursive and do not have chaining of learnt rules (i.e. `p :- q.  q :- r.` is a chain of learnt rules), then for a given label, we only need to use one rule to derive the label from the background knowledge. Therefore, we have the choice rule

```
1 { use(0..M) } 1.
```

where $\texttt{M} = |S_M^{\text{opt}}|$, so each of the generated answer sets corresponds directly to one of the rules in S_M^{opt}.

The answer sets of P_y will correspond to the choices of neural network labels and a rule that proves the task, then we can simply compute the semantic loss by Definition 5.1.1.

Remark 5.2.1. The propagation of the gradients back to the neural network is handled by Autograd mechanism in PyTorch [22]. The atoms present in each answer set are encoded in a bit mask tensor[1]. Since the answer sets will always be the same for a given label, this tensor can be cached on the GPU so the training stage can be efficiently performed on the GPU. This is a benefit of NeuralFastLAS over end-to-end training frameworks that rely on abductive learning mechanisms [18] since data must be moved constantly between the CPU and GPU, which may lead to performance bottlenecks [23].

To see how the semantic loss is computed, we must first understand the representation of the mask of an answer set:

Definition 5.2.2. Let A be an answer set and $\{X_i\}$ be a set N variables that we associate a probability with. The mask of A is denoted M_A and it is defined as

$$(M_A)_i = \begin{cases} 0 & \text{if } X_i \in A \\ 1 & \text{otherwise} \end{cases}$$

To see how this relates to the semantic loss, suppose we have the vector $\vec{P}$ which stores the probabilities, so $\vec{P}_i = P_\theta(X_i|x)$ if $X_i \in \mathcal{Z}$ and $\vec{P}_i = P(X_i|\theta_R)$ if $X_i \in S_M^{\text{opt}}$. Letting $|\cdot|$ denote the element-wise norm and starting with the product of all of the elements of $|M - \vec{P}|$, we can compute

$$\begin{aligned} \prod_i |M - \vec{P}|_i &= \prod_i \left|(M_A)_i - \vec{P}_i\right| \\ &= \prod_{X_i \in A} \left|0 - \vec{P}_i\right| \prod_{X_i \notin A} \left|1 - \vec{P}_i\right| \\ &= \prod_{X_i \in A} \vec{P}_i \prod_{X_i \notin A} 1 - \vec{P}_i \end{aligned}$$

to see that the product of the element of $|M - \vec{P}|$ corresponds exactly to the summand of the semantic loss in Definition 5.1.1.

With this, we can now define the algorithm with which the semantic loss is defined. The operations are defined in this way so that they can use operations that are compatible with PyTorch Autograd, which is important to allow the gradient to be propagated back.

[1] The idea of using the bit mask to make the semantic loss compatible with automatic differentiation is also used in https://github.com/UCLA-StarAI/Semantic-Loss/blob/master/semi_supervised/semantic.py

Algorithm 2 Compute semantic loss on a batch of data

Input: A batch B of n datapoints
  batch_loss = 0
  for $\langle x, y \rangle \in B$ **do**
    $l = 0$
    Run perceptron network on each raw data input $x_i \in x$ to get the vectors $P(z|x_i)$
    Let $\vec{P}$ be the concatenation of the vectors $[P(S_M^{\text{opt}}|\theta_R)$, $P(z|x_1)$, ..., $P(z|x_m)]$
    for $A \in \text{AS}(P_y)$ **do**
      Let M_A be the mask of A
      $L_A = |M_A - \vec{P}|$
      $l \mathrel{+}= \prod_{l_a \in L_A} l_a$
    end for
    batch_loss $\mathrel{+}= -\log l$
  end for
  return batch_loss$/B$

5.3 A Note on Rule Gradients and Learning Rate

In some tasks where the opt-sufficient subset is large, it is possible to create a negative feedback loop where the semantic loss favours incorrect rules: the network predicts incorrect labels, so the back-propagation of the semantic loss increase the probability of incorrect rules, which in turns means the next iteration of back-propagation will cause the network to converge towards incorrect predictions.

Intuitively, we can see the rules probabilities learnt to be, in some sense, a measure of coverage. For some tasks where the S_M^{opt} contains many subrules of the correct rule, these incorrect subrules may receive stronger signals by the semantic loss and leads to convergence into a local minimum.

To combat this, this report suggests two methods of using rule probabilities while training:

1. **Gradient Descent**: The method of gradient descent on the posterior rule distribution works best for tasks where the S_M^{opt} rules does not contain subrules of the correct rules. An example of where gradient descent on the rules performs well is analysed in section 6.1.
2. **Constant Distribution**: In cases where the opt-sufficient subset contains many subrules of the correct rules, it can be more effective to not perform gradient descent on the rule distribution and instead weight each rule equally or use the prior $P_\sigma(H|B)$ defined in Equation 4.3.

5.4 Inference

After the network has been trained and the final solving stage outputs the solution H, NeuralFastLAS can be used to perform inferences. A naive solution (where the perceptron model performs single label classification) would be to take the latent labels to be the elements that the network predicts with the highest score. However, there is a chance that the network predicts the wrong labels which violates some background constraints we may have on the domain (for example, suppose we have raw data inputs representing the

squares of a chessboard, we should not accept a prediction with two black kings on the board). Instead, we will use an ASP program to find the best neural network choice that do not violate any constraints we have in the background knowledge.

Let $x \in \mathcal{X}$ be the input raw data to perform an inference on. We run the trained network on each raw data input $x_i \in x$ to get the probabilities $p(z_j|x_i)$. As before, we use the log transformation to turn the maximization problem into a minimization one:

$$\max_{\substack{z \in \mathcal{Z} \\ z \text{ is valid}}} \prod_i P_\theta(z_j|x_i) = \min_{\substack{z \in \mathcal{Z} \\ z \text{ is valid}}} \sum_i - \log P_\theta(z_j|x_i)$$

We construct the inference program P_I with the following components:

1. The background knowledge B and choice rules that select the latent labels. For example, in single-label classification, the choice rules are

   ```
   1 { nn(1, z_1); ... nn(1, z_m) } 1.
   ...
   1 { nn(n, z_1); ... nn(n, z_m) } 1.
   ```

   for each of the raw data inputs $x_1, ..., x_n \in x$.

2. We invent a predicate `penalty/3` that will encode information about what probability the network predicted for each latent concept. For each atom representing a latent concept choice, we the term

   `penalty(`i`, `j`, `$-\log P_\theta(z_j|x_i)$`)`

3. Finally, we define the weak constraint to minimize over these penalties as

   ```
   :~ nn(i, z_j), penalty(i, j, P).[P@0, i, j]
   ```

Hence we are able to use the full power of symbolic reasoning during our inferences to potential catch any small inaccuracies in the network after training.

Chapter 6

Evaluation

This chapter evaluates the performance of the NeuralFastLAS architecture on a variety of tasks to evaluate its ability to learn both simple and complex tasks. To begin, we introduce the datasets that are evaluated:

1. **MNIST Arithmetric:** The MNIST arithmetic tasks have the following form: given three MNIST images, compute an answer by learning an arithmetic formula of the digit values. The formulae consist of addition and multiplication symbols. We use three types of MNIST Arithmetic datasets:
   (a) **MNIST Addition:** In this dataset, given three MNIST digits the final output is the sum of these three digits.
   (b) **MNIST Multiplication:** In this dataset, given three MNIST digits the final output is the product of these three digits.
   (c) **MNIST Mult-Add:** This is a new dataset where given three MNIST digits, the output is the product of the first two digits plus the final digit (i.e. $a, b, c \mapsto a \times b + c$).
2. **MNIST Chess:** The MNIST chess task consists of 3×3 chess boards and a model must learn to classify the state of the board (i.e. determine if the black king is safe, checkmated or the game is a draw). This dataset was introduced in [21][1]. We look at two versions of this dataset to discuss the strengths and limitations of the NeuralFastLAS algorithm.
   (a) **ChessMate:** In this task, the labels are one of: `safe`, `draw` and `mated_by(..)`, where the argument of the final label is a white piece. This version of the dataset allows for a very complicated search space and demonstrates the power of NeuralFastLAS.
   (b) **ChessState:** This is the version of the dataset from [21]. The labels are one of: `safe`, `draw` and `mate`. NeuralFastLAS is unable to train correctly on this dataset, but we explore it nonetheless to motivate ideas for future work.

   The MNIST chess task has never been tested on a true end-to-end neurosymbolic learner before. It is a very complex task give the size of the hypothesis space and rules learnt, we use it to demonstrate that NeuralFastLAS can perform well in complex tasks.

[1]The data for the Chess dataset is available at https://bitbucket.org/tsamoura/neurolog/src/master/data/chess/. Last accessed: 11/06/2022.

As far as the author of this report is aware, there exist only two other neurosymbolic models that train the neural and symbolic components simultaneously: $Meta_{Abd}$ [18] and NSIL [20]. The code for both of these architectures is private, so it is not possible to make a direct comparison. However, by demonstrating the power of NeuralFastLAS on the difficult MNIST Chess task, we illustrate that NeuralFastLAS successfully learns on tasks that are arguably more complex than what has been demonstrated in previous end-to-end learning benchmarks.

6.1 MNIST Arithmetic

There are no true dataset comparisons in the the evaluation of previous end-to-end neurosymbolic architectures [18, 20], however both have some form of arithmetic on MNIST images. NSIL [20] only learns addition and multiplication of two single-digit numbers. In this report, we demonstrate NeuralFastLAS on a task with a larger search space. For this reason, we will look at arithmetic on three single digit numbers since it is a more complicated learning task. Using three single-digit inputs to the arithmetic task is closer to the evaluation task used in $Meta_{Abd}$ [18]. Each of the MNIST Arithmetic datasets have 25,000 training examples, 5,000 validation examples and 5,000 training examples so that the dataset size is similar to [20].

Since the code for $Meta_{Abd}$ and NSIL is private, we cannot use them as baseline comparisons. Instead, we will compare to various fully neural architectures:

1. **CNN:** The CNN is the most simple of the fully neural models used. The input is of size $3 \times 28 \times 28$ where each channel represents one input images.
2. **ResNet-18:** ResNet-18 [24] is a deep convolutional neural network. It is a very powerful image recognition network, but we show in this report that is struggles to transfer to reasoning tasks.
3. **Concept Bottleneck Model:** The Concept Bottleneck Model [25] is a two component model: the first component is a CNN which takes each input image as input and returns a latent encoding. The output vectors from the first component are concatenated and forwarded through an MLP which acts as the reasoning component that outputs a final answer.
4. **Neural Arithmetic Logic Unit:** The NALU [26, 27] is a neural component designed specifically to learn arithmetic functions[2]. It is a recurrent network - the three inputs are sequentially fed to the model.

The complete details for the datasets and each of the neural models, as well as the mode bias for NeuralFastLAS, can be found in Appendix B. For the NeuralFastLAS evaluations, the training stage is run for only one epoch, since this is enough to achieve sufficiently high accuracy for solving. The correct solution is learnt in each task and is given by:

```
MNIST Add:      f(V0,V1,V2,V3) :- add(V0,V1,V4), add_acc(V4,V2,V3).
MNIST Mult:     f(V0,V1,V2,V3) :- mult(V0,V1,V4), mult_acc(V4,V2,V3).
MNIST Mult-Add: f(V0,V1,V2,V3) :- mult(V0,V1,V4), add_acc(V4,V2,V3).
```

[2]The implementation used it provided by the authors of [27] and can be found at https://github.com/AndreasMadsen/stable-nalu. Last accessed 18/06/2022.

After solving, the network is fine-tuned using only the final hypothesis for a further nine epochs. While fine-tuning for nine epochs is not necessary, it allows us to make some comparisons between the number of epochs needed to train NeuralFastLAS relative to the fully neural models, as can be seen in Figure 6.1.

	Performance on held out test set		
	MNIST Add	**MNIST Mult**	**MNIST Mult-Add**
CBM	88.1%	95.7%	94.3%
ResNet-18	80.2%	86.9%	87.7%
CNN	48.1%	67.9%	53.1%
NALU	0.03%	0.00%	0.19%
NALU†	32.2%	0.12%	0.46%
NeuralFastLAS	**96.8%**	**97.0%**	**97.1%**

Table 6.1: Performance of various models on the MNIST Arithmetic tasks. †One version of the NALU was trained on 55000 training examples and 1000 epochs since RNNs typically require large datasets.

From Table 6.1, we can see that NeuralFastLAS surpasses any of the fully neural models. While the CBM performs reasonably well in all tasks, NeuralFastLAS shows that true symbolic reasoning is more powerful than an MLP reasoning component. It is also instructive to note that the performance of the CBM is approximately equal to the performance of NeuralFastLAS on the training data for MNIST-Mult (see Figure 6.1), however the CBM does not generalise as well to the test set.

Interestingly, the NALU performs poorly in both the normal and enlarged dataset. This may be since RNNs are difficult to train due to issues with propagating gradients back [28]. The evaluation in [27] has the cumulative sum and product as labels (so for the ten-long sequence of digits used in the paper, there would be ten labels as opposed to one), so it is able to propagate more information back. This may explain the higher performance in [27] than in this dataset, where only the final label is propagated back. We show an advantage of ILP since it requires significantly less data than fully neural methods [2].

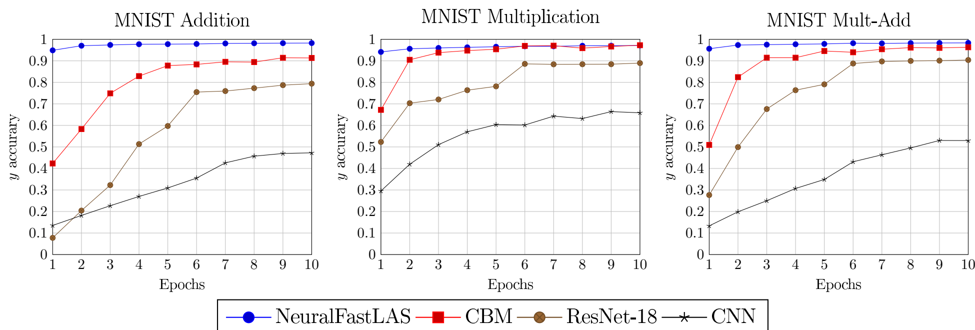

Figure 6.1: Performance of various models on the MNIST Arithmetic Tasks on the training data. NALU is not included since its performance is poor after only 10 epochs.

Furthermore, it is worth comparing how the each model performs across the tasks. Most of the fully neural models perform worse on the addition task than the other tasks. This may be due to the sparsity of the downstream labels: in the addition task, the frequency distribution of labels is very dense, whereas in the multiplication task the distribution of the downstream labels is much more sparse. The NALU performs much worse on the tasks involving multiplication, this is because the NALU differs from the other neural models in

that it solves the task as a regression rather than classification, so it may favour a denser frequency. On the hand, the accuracy of NeuralFastLAS is reasonably consistent across the tasks - this demonstrates how the symbolic component is more powerful at reasoning about discrete structures (e.g. the integers) than a fully differentiable model and the technique is more readily transferable between similar tasks.

6.1.1 Learning the Posterior Rule Distribution

We will also take the opportunity to look at how the rule distribution changes during the NeuralFastLAS training phase. For this example, we will focus on the MNIST Mult-Add example due to some interesting symmetries in the rules learnt.

Looking at Figure 6.2, we see that initially, every rule has the same probability. It is interesting to see that the only rules whose probabilities grow are of the form $a \times b + c$, $a \times c + b$ and $b \times c + a$. This is likely since all of these rules produce the same distribution of labels so are equally probable as explanations when the network is untrained. However, when trained, the network starts to favour the correct rule and we see it quickly become the dominating rule with the first quarter of the epoch.

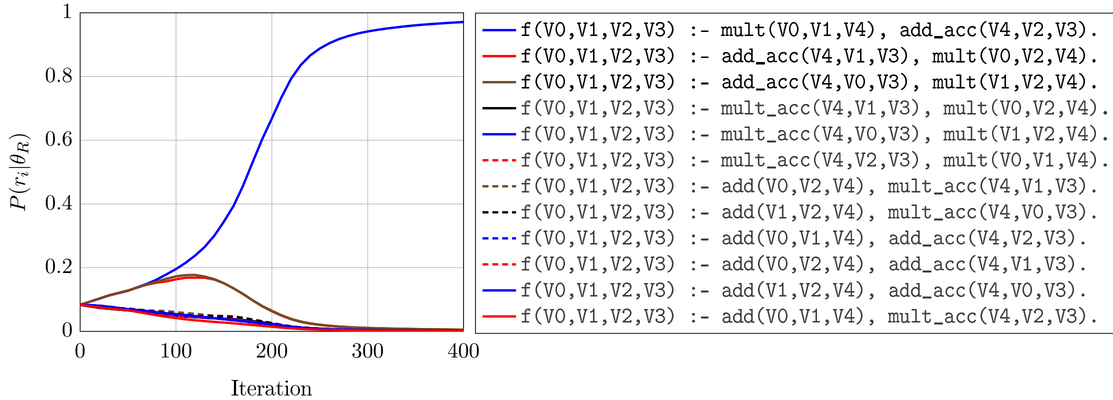

Figure 6.2: The learnt posterior distribution of rules on the MNIST Mult-Add task for the first quarter of the first epoch.

To understand why the other rules decrease in probability, consider a data point with final label 47 (for example, this could be generated by $6 \times 7 + 5$). The rule $a + b + c$ will never prove this since $a + b + c \leq 27$. Additionally, $a \times b \times c$ cannot prove this since 47 is prime. After computing the semantic loss, the gradient propagated back will decrease the scores of these rules. So any rule that produced a different distribution of labels to the actual task is likely to be unable to prove some of the labels in each batch which causes its score to decrease in each gradient step.

The power of learning a rule distribution is that NeuralFastLAS is able to rule out rules that cannot explain every example and hence is essentially able to fine-tune on the correct rule before even reaching the solving stage. This leads to better predicted probabilities to use in solving.

6.2 MNIST Chess

The MNIST Chess dataset is a collection of 9,000 training examples and 9,000 test examples. Each example consists of a 3x3 chess board represented by MNIST digits. Details on the representation of the dataset can be found in section B.3.

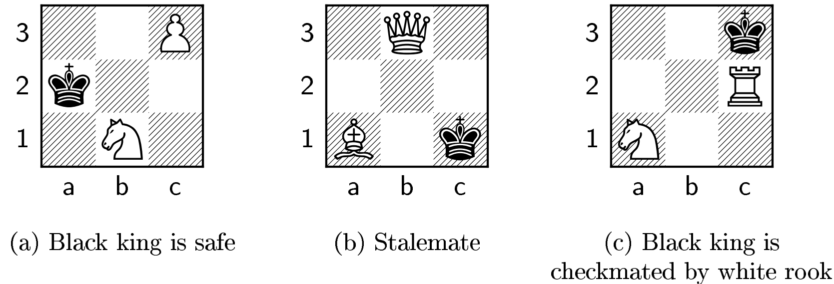

Figure 6.3: Examples from the MNIST Chess dataset

The board always consists of three pieces: the black king, and two distinct white pieces. With these constraints, there are exactly 7560 possible chessboard states available. The power of the NeuralFastLAS abduction algorithm means that we only consider the valid combinations of neural network outputs, rather than all $8^9 = 134,217,728$ possible combinations. We encode information about the board layout via constraints in the background knowledge.

Note that this dataset is particularly challenging for a multitude of reasons, partially due to the complexity of rules learnt, but also due to the unbalancedness of the dataset. From Figure 6.4, there are a couple of observations to make: firstly, the dataset is dominated by blank spaces - indeed 6/9 squares of each board is blank. Only $2/9 \approx 22\%$ of the dataset corresponds to white pieces, and some of these are incredible infrequent, for example the white knight and the white pawn.

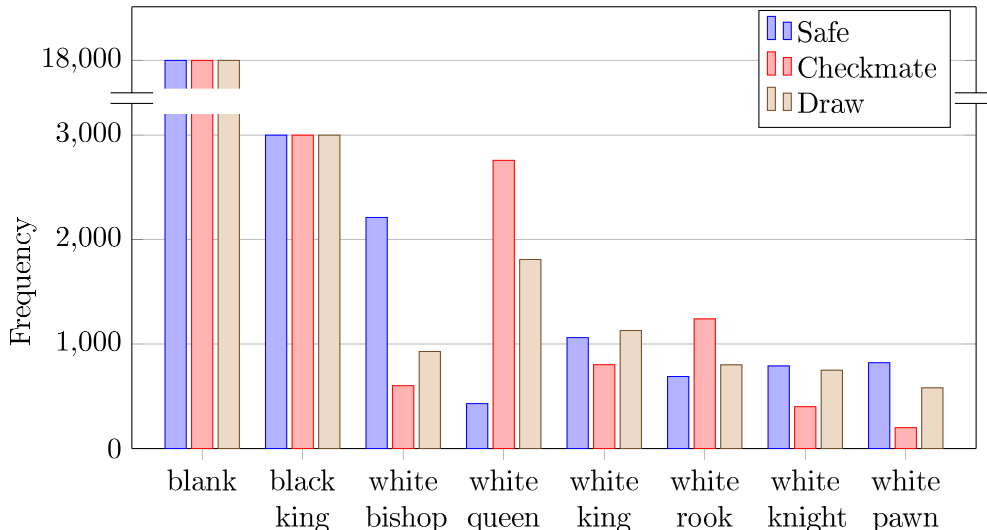

Figure 6.4: Frequency of pieces per final game state in the Chess dataset

The uneven distribution of pieces is caused by the nature of chess - for example, a white

pawn can never deliver checkmate on a 3×3 board with only one other white piece, so it appears very infrequently in the set of checkmate boards (for example, a case where the white pawn may be present in a checkmate is when the queen is delivering the checkmate and the pawn is guarding the queen). The same is true for the white knight.

Remark 6.2.1 (A Note on the Mode Declaration)**.** In both tasks, the decision was made *not* to reduce the search space to only concern the black king since this makes the task significantly more challenging, but shows that NeuralFastLAS still learns in some cases with an extremely large search space.

6.2.1 ChessMate Task

In this task, the correct solution is:

```
safe         :- can_move(b_king).
draw         :- not can_move(b_king), not is_attacked(b_king).
mated_by(WP) :- not can_move(b_king), attacks(WP, b_king).
```

The learning task is produced by taking a subset of the dataset that contains 36 examples divided equally between safe states, draws and checkmates. The S_M^{opt} set for this example contains 187 rules, which is significantly larger than any of the arithmetic tasks.

The network is trained for 10 epochs, then the solving stage of NeuralFastLAS is invoked to learn the rules, and finally the network is fine-tuned with the correct rules for a further 10 epochs. From the confusion matrix in Figure 6.5, we can see that the network is able to accurately identify blank squares, the black king, and the white queen. This is expected since these are most frequently occurring in the dataset.

Target Class \ Predicted Class	blank	b. king	w. rook	w. bishop	w. knight	w. king	w. pawn	w. queen
blank	100							
b. king		98						
w. rook	1	1	91	2	1			1
w. bishop				90	6			1
w. knight			35		59			2
w. king					98			
w. pawn	1		84		13			
w. queen								97

Figure 6.5: Confusion Matrix for the ChessMate Task. Values represent percentages, each class was tested on 1000 examples.

The network is unable to accurately predict the white king and the pawn. This is most likely due to the fact that these two pieces can never deliver checkmate, so there are no labels of the form `mate_by(white_king)` nor `mate_by(white_pawn)`. The mate labels

are likely where the semantic loss is able to propagate back the strongest signals since the number of combinations of pieces that explain a specific checkmate is limited - in particular, the piece that delivers the checkmate must be in the answer set.

Despite the inaccuracy that the trained NeuralFastLAS algorithm has in predicting some of the white pieces, it is still able to successfully learn the correct rules. Negative examples [7] are used to ensure that the state of every board is unique (for example, it is not possible for a chessboard to both be safe and a draw).

Table 6.2 shows the performance of various fully-neural architectures relative to various fully neural networks. As before, the CBM performs reasonably well, whereas ResNet-18 does not. This is likely because the CBM has many linear layers at the end which act as a reasoning component whereas ResNet-18 only has one linear layer, so its ability to model reasoning is limited. We see that NeuralFastLAS still performs better than any fully neural model, and is able to achieve very high accuracy on the latent labels for the chess pieces once it is fine-tuned.

	Performance on held out test set	
	Latent Accuracy	**ChessState**
CBM	-	97.3%
ResNet-18	-	76.1%
CNN	-	70.6%
NeuralFastLAS	99.5%[3]	**98.3%**

Table 6.2: Performance of Various Models on the MNIST Chess Tasks. Only NeuralFast-LAS has a latent accuracy since the latent representation is semi-supervised. The CBM model learns an incomprehensible latent representations in its bottleneck.

6.2.2 ChessState Task

The ChessState task is a more difficult for NeuralFastLAS than the ChessMate task. This is because the labels of the chess boards are more general so less distinguishing information is propagated back to the network by the semantic loss function.

The confusion matrix in Figure 6.7 shows the predictions of the neural network after the NeuralFastLAS training stage. The network is very accurate in identifying the blank squares and the black kink, this is likely since these pieces account for a majority of the dataset and so receive strong signals during every back-propagation.

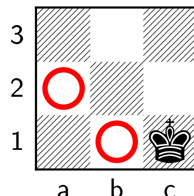

Figure 6.6: Example of checkmate situation with many possible labels with white pieces

[3]Note that this value is accuracy on the unbalanced chess dataset, not the balanced MNIST dataset - hence the very high accuracy for a simple CNN

However, the network is a very poor predictor of white pieces. This is understandable since the labels do not contain much information about the pieces. To understand how this differs from the ChessMate task, consider the example in Figure 6.6 and suppose that it has the label `mate`. The network is confident in predicting the blank spaces and blank king, so the white pieces are likely to be in the red regions in this example.

Consider first only the correct checkmate rule, then combinations of the white pawn, white rook, white queen and white king satisfy the checkmate label. Many checkmates are delivered by the queen, and it is generally paired with a bishop or the king, so the network has learnt to mispredict the partner of the white queen on the board as the white queen.

We can compare this to the ChessMate label: if we had the label `mated_by(white_rook)` then we narrow down the possible combinations to exactly one (white rook on b1 and white queen on a2), so the semantic loss is much more informative. Now if we add back into consideration the fact that the semantic loss is also finding solutions using incorrect rules, the signal from the correct answer set is very weak (since there are many possible answer sets) and this leads the network to get stuck in a local minimum.

Target Class \ Predicted Class	blank	b. king	w. rook	w. bishop	w. knight	w. king	w. pawn	w. queen
blank	100							
b. king		99						
w. rook	2	1		92	1			2
w. bishop	1			1	2			94
w. knight	1			93	4			
w. king				3				95
w. pawn	2			97				
w. queen				1	98			

Figure 6.7: Confusion Matrix for the ChessState Task. Values represent percentages, each class was tested on 1000 examples.

Chapter 7

Related Work

As discussed in chapter 1, deep learning has a number of pitfalls. Interest has therefore been directed towards seeking inspiration from traditional rule-based symbolic AI whose strengths match the weaknesses of deep learning - in particular, interpretability and data efficiency. As a result, the field of neurosymbolic machine learning as seen a variety of new architectures in recent years.

The recent work in neurosymbolic machine learning can be roughly grouped into four categories:

1. Frameworks that inject symbolic logic into pure differentiable architectures with the objective of enhancing them with the capability of reasoning while maintaining a fully differentiable model. Examples of these systems include Logic Tensor Networks [29] and Logical Neural Networks [30]. Logic tensor networks inject logical constraints over relations about objects to constrain the training of a deep neural network, while logical neural networks inject symbolic relations directly into the neurons of the network to simultaneously provide learning and symbolic reasoning.

2. The second category of the symbolic systems use symbolic logic and reasoning as a "controller" over differentiable modules. Symbols are represented in a vector space and their embeddings are learnt during a training process based on Prolog-like backward chaining. An example of such approaches in the Neural Theorem Prover [31] which embeds logical symbols into a sub-symbolic vector representation and uses backward chaining to construct neural networks and radial basis function kernels for learning such vector representations. Symbols with similar meaning are "close together" in the learnt vector representation

3. In contrast, the third stream of neurosymbolic machine learning consists of approaches that implement logical inference is a fully differentiable architecture to generate data efficient inductive systems that can learn explicit human-readable symbolic rules. Examples of such frameworks are δILP [32] and the Neural Logic Programming framework [33].

4. The fourth and final group of neurosymbolic models are those that combine neural and symbolic components while keeping a clear distinction between the two. Low-level, unstructured data is processed by a neural component who outputs are used by a symbolic inference component capable of performing deductive inference or inductive rule learning. Examples include the sequence neurosymbolic learning framework

FF-NSL [34] which learns general and interpretable symbolic knowledge and is capable of solving classification tasks with a higher accuracy than fully differentiable models.

There are other architectures with distinct neural and symbolic components that fit into this category although they are not truly end-to-end, often since they require hand engineered rules and are not capable of learning rules jointly with training the network. Examples of such systems are DeepProbLog [35] and NeurASP [36]. The emphasis on these approaches is an data efficiency, while preserving a high-level of accuracy in the differentiable neural component.

NeuralFastLAS, NSIL [20] and $Meta_{Abd}$ [18] are also included in the fourth category of neurosymbolic models, but these three models are able to train the neural and symbolic components jointly.

In short, NSIL works by iteratively solving a Learning from Answer Sets task (using either FastLAS or ILASP), and then training a neural network for one epoch using NeurASP. In the first iteration, the learning from answer sets task is created by bootstrapping with a set of examples that covers every target label. After the first epoch, the examples are modified using information about coverage of the hypothesis and predictions for the neural network.

Since the scoring function of NeuralFastLAS is motivated by the derivations in [18], we explore the architecture in detail.

7.1 The $Meta_{Abd}$ Architecture

The $Meta_{Abd}$ architecture [18] builds on the mechanism of meta-interpretive learning [37] to use abduction and induction to train a neural network in a neurosymbolic end-to-end manner. This work introduces some important mathematical formalisations of end-to-end neurosymbolic training will be necessary to understand at detail.

In abductive meta-interpretive learning, if we suppose the dataset $\mathcal{D}$ is i.i.d. sampled from the underlying distribution, then the objective is given by

$$(H^*, \theta^*) = \arg\max_{H,\theta} \prod_{\langle x,y\rangle \in \mathcal{D}} \sum_{z \in \mathcal{Z}} P(y, z, |B, x, H, \theta) \tag{7.1}$$

In $Meta_{Abd}$, it is not possible to optimize the hypothesis H with the neural parameters θ since $\mathcal{H}$ is a discrete space and Θ is a continuous space. Instead, $Meta_{Abd}$ proposes to treat H like z as a latent concept and solve the optimisation problem

$$\theta^* = \arg\max_{\theta} \prod_{\langle x,y\rangle \in \mathcal{D}} \sum_{H \in \mathcal{H}} \sum_{z \in \mathcal{Z}} P(y, H, z, |B, x, \theta) \tag{7.2}$$

By repeated application of Bayes' Rule, one can rewrite the probability in the above equation as

$$\begin{aligned} P(y, H, z|B, x, \theta) &= P(y, H|B, z)P_\theta(z|x) \\ &= P(y|B, H, z)P(H|B, z)P_\theta(z|x) \\ &= P(y|B, H, z)P_{\sigma^*}(H|B)P_\theta(z|x) \end{aligned}$$

where $P_{\sigma^*}(H|B)$ is the Bayesian prior distribution on first-order logic hypothesis, in [18], this is defined as

$$P_{\sigma^*}(H|B) = \frac{6}{\pi^2|H|^2} \tag{7.3}$$

and the probability $P(y|B, H, z)$ is given by

$$P(y|B, H, z) = \begin{cases} 1 & \text{if } B \cup H \cup z \models y \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

With this formulation of the problem, $Meta_{Abd}$ attempts searches for a hypothesis H and then abduces possible pseudolabels z that satisfy $H \cup B \cup z \models y$. In particular, each combination of H and z found are scored according to

$$\text{score}(H, z) = P_{\sigma^*}(H|B)P_\theta(z|x)$$

since by abduction, we are already have that for any pair H, z found, $H \cup B \cup z \models y$. The abduction mechanism is added to the deducion and induction mechanisms already present in Metagol [38].

The $Meta_{Abd}$ pipeline trains the model with the following steps:

1. Given a batch of data $\{\langle x_i, y_i\rangle\}_{i\in B}$, perform a forward pass of the neural network to get a the batch of output vectors $\{\hat{z}_i\}_{i\in B}$ where $\hat{z}_{i,k} = P_\theta(\hat{z}_k|x_i)$ (or more simply put, the neural network outputs for each data point the predicted possibilities of each latent label being the true label).

2. The $Meta_{Abd}$ algorithm then finds $H, \{z_i\}_{i\in B}$ that satisfies

$$(H', \{z_i\}_{i\in B}) = \underset{H,\{z_i\}_{i\in B}}{\arg\min} \sum_{i\in B} score(H, z_i)$$

   Please note that it is not actually clear how the $Meta_{Abd}$ algorithm reduces scores of a batch to a single score, but for the sake of demonstration it is sufficient to assume that the scores are simply summed. In fact, it is not truly clear from the algorithm that $Meta_{Abd}$ learns a hypothesis from a batch at a time; the algorithm in [39] seems to suggest that each datapoint learns a hypothesis independently.

3. The $\{z_i\}_{i\in B}$ are then treated as the true latent labels to the neural network and the error is then propagated back through the network via whichever loss function was chosen for the task.

While this work is very novel as possibly the closest work to end-to-end neurosymbolic learning, it has a couple of drawbacks worth acknowledging:

1. The search for the optimal hypothesis of a batch and the latent labels essentially is a greedy depth first search due to the semantics of Prolog

2. The hypothesis learnt is not maintained between batches - the hypothesis is learnt from scratch between epochs, which means the algorithm does not utilize any information from the history of learning.

7.2 Semantic Loss

The Semantic Loss function [5] is a differentiable loss function that captures how close a neural networks outputs are to satisfying a set of constraints. The intuition is that it assumes a Bernoulli distribution over the predicted probability of a set of atoms and then computes a value that roughly relates to the probability that an answer set satisfies a given constraint.

Definition 7.2.1 (Semantic Loss Function [5]). Let $X = \{X_1, ..., X_n\}$ be a set of atoms and p be a vector of probabilities and p_i represents the probability of X_i, and corresponds to a single output of the neural network. If α represents a sentence in propositional logic, then we define the semantic loss as

$$L^s(\alpha, p) \propto \log \sum_{x \models \alpha} \prod_{\substack{i \in \{0,...,n\} \\ x \models X_i}} p_i \prod_{\substack{i \in \{0,...,n\} \\ x \models \neg X_i}} (1 - p_i)$$

Chapter 8

Conclusion

The main contribution of this work has been providing a novel end-to-end neurosymbolic learning architecture. NeuralFastLAS makes guarantees on the optimality of the opt-sufficient subset produced and this report demonstrates a sufficient condition on the neural network training to guarantee the correct final solution. NeuralFastLAS is often able to find the correct solution even when the neural network is not trained well, as we saw in the evaluation of the ChessMate task in subsection 6.2.1.

We have shown that NeuralFastLAS performs better than fully neural methods on a variety of datasets that require the model to perform some reasoning, such as learning arithmetic or the rules of chess. We have also identified a side effect of using the semantic loss for training a network in terms of causing the network to get trapped in local minima. Some ideas are discussed in the next section for future work on investigating this further.

End-to-end training of neurosymbolic architectures is a relatively unexplored field within machine learning. It is able to leverage the advantages of both neural networks and symbolic learners, capturing the "best of both worlds", to learn complex tasks and generalise well on test datasets. More work in this area will certainly lead to some very impressive results.

8.1 Future Work

The issue of models becoming easily stuck in local minima is not very well documented in current neurosymbolic literature. NeuEx [40] explicitly runs its search algorithms multiple times with different initialization parameters in order to try to avoid local minima. However, it will be constructive to explore more sophisticated ways of reduce the impact of local minima.

Exploration should be done to understand the loss landscape that arises from the use of the semantic loss. To motivate further ideas, we quickly turn our attention to Sentential Decision Diagrams [41]. SDDs can be used as a tractable representation for boolean functions. For our use, we can represent the semantic loss function as an SDD which will allow us to reason about the loss function with a broader array of tools.

It is the author's opinion that the binary nature of the outputs of relations $\vee$ and $\wedge$ create highly non-convex loss landscapes, but a more formal investigation needs to be done to

understand this. Differentiable logic machines [42] show success in introducing annealing to the logic gates, which lessens the effect of the "all-or-nothing" nature of logic operations.

Further exploration to the nature of symbolic loss functions to find ways of "smoothing" the loss landscape will lead to better convergence of the neural network in the training stage of NeuralFastLAS, which in turns leads to better predictions for the final solving stage.

Another interesting avenue of exploration would be to explore the use of various deep learning models in NeuralFastLAS for more complex tasks. For example, auto-encoders have been used to perform feature extraction in the past [43, 44].

One immediate use of auto-encoders in NeuralFastLAS would be as a pre-processing step: an auto-encoder could be trained to map the raw data to some latent space of a smaller dimension than the raw data. With this, the network trained by NeuralFastLAS would be a separate, shallow network which learns a clustering of the latent space of the auto-encoder. This reduces the complexity of the training stage of NeuralFastLAS since much of the work of feature extraction has already been done by the pre-trained auto-encoder. This will allow the NeuralFastLAS to scale easily as the unstructured data becomes more complex.

Bibliography

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

[2] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *CoRR*, abs/1804.11285, 2018. URL http://arxiv.org/abs/1804.11285.

[3] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, 23–24 Feb 2018. URL https://proceedings.mlr.press/v81/buolamwini18a.html.

[4] Christina Pazzanese. Great promise but potential for peril: Ethical concerns mount as ai takes bigger decision-making role in more industries. *The Havard Gazette*. URL https://news.harvard.edu/gazette/story/2020/10/ethical-concerns-mount-as-ai-takes-bigger-decision-making-role/.

[5] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *CoRR*, abs/1711.11157, 2017. URL http://arxiv.org/abs/1711.11157.

[6] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer Berlin Heidelberg, 2003. ISBN 9783540006787. URL https://books.google.co.uk/books?id=VjHk2Cjrti8C.

[7] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Fastlas: scalable inductive logic programming incorporating domain-specific optimisation criteria. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2877–2885, 2020.

[8] Vladimir Lifschitz. Twelve definitions of a stable model. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, pages 37–51, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-89982-2.

[9] Alessandra Russo and Mark Law. Lecture notes for the logic based learning course at imperial college london.

[10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *CoRR*, abs/1705.09811, 2017.

[11] Mark Law, Alessandra Russo, and Krysia Broda. The ILASP system for learning answer set programs. www.ilasp.com, 2015.

[12] Mark Law, Alessandra Russo, Krysia Broda, and Elisa Bertino. Scalable non-observational predicate learning in asp. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1936–1943. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/267. URL https://doi.org/10.24963/ijcai.2021/267. Main Track.

[13] Mark Law, Alessandra Russo, Krysia Broda, and Elisa Bertino. Proofs of the theorems in scalable non-observational predicate learning in asp, 2021. URL https://github.com/spike-imperial/FastLAS/blob/master/fast_non_opl_proofs.pdf. Last Accessed: 03/06/2022.

[14] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/9c19a2aa1d84e04b0bd4bc888792bd1e-Paper.pdf.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[16] Oliver Ray, K. Broda, and Alessandra Russo. Hybrid abductive inductive learning: A generalisation of progol. volume 2835, pages 311–328, 09 2003. ISBN 978-3-540-20144-1. doi: 10.1007/978-3-540-39917-9_21.

[17] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In Stephen H. Muggleton, Alireza Tamaddoni-Nezhad, and Francesca A. Lisi, editors, *Inductive Logic Programming*, pages 91–97, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31951-8.

[18] Wang-Zhou Dai and Stephen Muggleton. Abductive knowledge induction from raw data. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1845–1851. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/254. URL https://doi.org/10.24963/ijcai.2021/254. Main Track.

[19] Céline Hocquette and Stephen Muggleton. How much can experimental cost be reduced in active learning of agent strategies? In Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese, editors, *Inductive Logic Programming*, pages 38–53, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99960-9.

[20] Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. Inductive learning of complex knowledge from raw data, 2022. URL https://arxiv.org/abs/2205.12735.

[21] Efthymia Tsamoura and Loizos Michael. Neural-symbolic integration: A compositional perspective, 2020. URL https://arxiv.org/abs/2010.11926.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[23] Yusuke Fujii, Takuya Azumi, Nobuhiko Nishio, Shinpei Kato, and Masato Edahiro. Data transfer matters for gpu computing. pages 275–282, 12 2013. ISBN 978-1-4799-2081-5. doi: 10.1109/ICPADS.2013.47.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[25] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. *CoRR*, abs/2007.04612, 2020. URL https://arxiv.org/abs/2007.04612.

[26] Andrew Trask, Felix Hill, Scott E. Reed, Jack W. Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. *CoRR*, abs/1808.00508, 2018. URL http://arxiv.org/abs/1808.00508.

[27] Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gNOeHKPS.

[28] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

[29] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *CoRR*, abs/2012.13635, 2020. URL https://arxiv.org/abs/2012.13635.

[30] Ryan Riegel, Alexander G. Gray, Francois P. S. Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, and Santosh K. Srivastava. Logical neural networks. *CoRR*, abs/2006.13155, 2020. URL https://arxiv.org/abs/2006.13155.

[31] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *CoRR*, abs/1705.11040, 2017. URL http://arxiv.org/abs/1705.11040.

[32] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *CoRR*, abs/1711.04574, 2017. URL http://arxiv.org/abs/1711.04574.

[33] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base completion. *CoRR*, abs/1702.08367, 2017. URL http://arxiv.org/abs/1702.08367.

[34] Daniel Cunnington, Mark Law, Alessandra Russo, and Jorge Lobo. FF-NSL: feed-forward neural-symbolic learner. *CoRR*, abs/2106.13103, 2021. URL https://arxiv.org/abs/2106.13103.

[35] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[36] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/243. URL https://doi.org/10.24963/ijcai.2020/243. Main track.

[37] Stephen H. Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. pages 1551–1557, 2013. URL http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6637.

[38] Andrew Cropper and Stephen H. Muggleton. Metagol system. https://github.com/metagol/metagol, 2016. URL https://github.com/metagol/metagol.

[39] Wang-Zhou Dai and Stephen H. Muggleton. Abductive knowledge induction from raw data. *CoRR*, abs/2010.03514, 2020. URL https://arxiv.org/abs/2010.03514.

[40] Shiqi Shen, Soundarya Ramesh, Shweta Shinde, Abhik Roychoudhury, and Prateek Saxena. Neuro-symbolic execution: The feasibility of an inductive approach to symbolic execution. *CoRR*, abs/1807.00575, 2018. URL http://arxiv.org/abs/1807.00575.

[41] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, page 819–826. AAAI Press, 2011. ISBN 9781577355144.

[42] Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, Paul Weng, Jianye Hao, Dong Li, and Wulong Liu. Differentiable logic machines. *CoRR*, abs/2102.11529, 2021. URL https://arxiv.org/abs/2102.11529.

[43] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/2a38a4a9316c49e5a833517c45d31070-Paper.pdf.

[44] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017. doi: 10.1109/IJCNN.2017.7965877.

[45] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). URL https://eur-lex.europa.eu/eli/reg/2016/679/oj. Last Accessed: 21/06/2022.

Appendix A

Ethical Considerations

The fields of machine learning and artificial intelligence are used in a wide variety of critical applications, hence it is important to acknowledge that machine learning algorithms may have faults such as unintended bias, a discussion of this is provided in [3].

However, NeuralFastLAS has greater interpretability than fully neural models which in turns leads to more principled ethical evaluations of the decisions made for a specific task. The rules learnt can be inspected and easily understood by humans, which helps align the use of AI with ethics regulations such as GDPR [45].

None of the datasets used in this report include any personally identifying information.

Appendix B

Evaluation Details

B.1 Baseline Models

In this section, we give definitions of all of the baseline models used:

1. **CNN:** The CNN was a simple CNN with two convolutional layers with kernel size of 5, each one followed by a max pooling layer. The second half of the network was three linear layers of sizes 120, 84 and an output size depending on the range of the final labels for the task. The intermediate layers all have ReLU activation functions, and the final layer has a softmax activation. For this model, the images are concatenated along the first axis and then inputted to the network, so each input channel represents an image.

2. **CBM:** The CBM has two components: first, a perceptron CNN - like the baseline CNN, it has two convolutional layers with kernel size of 5, each following by a max pooling layer, followed by linear layers of size 256 and 10 (or 8 for the Chess task). The second component of the CBM is an MLP with linear layers of size 256 and 256. The output size is is depending on the task.

3. **ResNet-18:** The ResNet-18 model used is provided by the `torchvision` package. The number of input channels was 3 for the MNIST Arithmetic tasks, and 9 for the Chess task. The number of outputs was changed depending on the task. PyTorch provides a tutorial on how to change the number of classes.[1]

4. **NALU:** The NALU code used is provided by [27]. The source code provides an MNIST arithmetic example which was modified to be exactly like the MNIST arithmetic task used for the evaluation of NeuralFastLAS.[2]

[1]This tutorial can be found at `https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html`. Last accessed 20/02/2022.

[2]The exact file from the repo used can be found at `https://github.com/AndreasMadsen/stable-nalu/blob/master/experiments/sequential_mnist.py`. Last accessed 18/06/2022.

Task	Number of CNN Outputs
MNIST Add	28
MNIST Mult	730
MNIST Mult-Add	90
ChessMate	8

Table B.1: Size of the output layer for the CNN, CBM and ResNet-18 for each task

B.2 MNIST Arithmetic Task

The mode bias for each of these tasks is defined as

$$M_h = \big(\ \texttt{f(var(-nn_label), var(-nn_label), var(-nn_label), var(+num)).}\ \big)$$

$$M_b = \begin{pmatrix} \texttt{add(} & \texttt{svar(+nn_label),} & \texttt{svar(+nn_label),} & \texttt{var(-num)).} \\ \texttt{add_acc(} & \texttt{var(+nn_label),} & \texttt{var(+num),} & \texttt{var(-num)).} \\ \texttt{mult(} & \texttt{svar(+nn_label),} & \texttt{svar(+nn_label),} & \texttt{var(-num)).} \\ \texttt{mult_acc(} & \texttt{var(+nn_label),} & \texttt{var(+num),} & \texttt{var(-num)).} \end{pmatrix}$$

So that `add/3` and `mult/3` represent the addition and multiplication of two neural network outputs, and `add_acc/3` and `mult_acc/3` represents the addition and multiplication of a neural network output with a result from a previous computation. `mult(5, 3, 8)` and `add_acc(4, 8, 32)` are examples of the use of these predicates.

The symbol `svar` in the mode bias denotes a symmetric argument as described in subsection 4.3.1.

B.3 Chess Dataset

MNIST Digit	Chess Piece
0	Empty
1	Black King
2	White Rook
3	White Bishop
4	White Knight
5	White King
6	White Pawn
7	White Queen

Table B.2: Mapping between MNIST digits and chess pieces in the chess dataset

The mode declaration for the ChessMate task is given by:

$$M_h = \big(\ \texttt{state(safe).}\quad \texttt{state(draw).}\quad \texttt{mate_by(var(white_piece)).}\ \big)$$

$$M_b = \begin{pmatrix} \texttt{can_move(const(piece))} \\ \texttt{not can_move(const(piece))} \\ \texttt{is_attacked(const(piece))} \\ \texttt{not is_attacked(const(piece))} \end{pmatrix}$$

The mode declaration for the ChessState task is given by:

$$M_h = \left(\texttt{state(safe).} \quad \texttt{state(draw).} \quad \texttt{state(mate).} \right)$$

$$M_b = \left(\begin{array}{l} \texttt{can_move(const(piece))} \\ \texttt{not can_move(const(piece))} \\ \texttt{is_attacked(const(piece))} \\ \texttt{not is_attacked(const(piece))} \\ \texttt{placed(const(piece), square(var(index), var(index)))} \\ \texttt{attacked(var(white_piece), square(var(index), var(index)))} \end{array} \right)$$