A Neurally-Guided, Parallel Theorem Prover

Michael Rawson[0000-0001-7834-1567] and Giles Reger

University of Manchester, UK
michael@rawsons.uk, giles.reger@manchester.ac.uk

Abstract. We present a prototype of a neurally-guided automatic theorem prover for first-order logic with equality. The prototype uses a neural network trained on previous proof search attempts to evaluate subgoals based directly on their structure, and hence bias proof search toward success. An existing first-order theorem prover is employed to dispatch easy subgoals and prune branches which cannot be solved. Exploration of the search space is asynchronous with respect to both the evaluation network and the existing prover, allowing for efficient batched neural network execution and for natural parallelism within the prover. Evaluation on the MPTP dataset shows that the prover can improve with learning.

Keywords: ATP · Graph Convolutional Network · Tableaux · MCTS

1 Introduction

Recent advances in neural network systems allow for processing graph-structured data in a neural context. Graphs are a natural representation for logical formulae as found in automatic theorem provers (ATPs), suggesting a new breed of *neural ATP* in which proof search is guided by a neural black-box acting as "mathematician's intuition". However, in practice there are several implementation issues [31] which must be avoided in order for neural systems to integrate with efficient traditional ATPs:

1. Proof state in such systems may be of impractical size, such as in saturation-based provers, leading to training data which is impractical to learn from and slow to evaluate. In a saturation context, the size of the current proof state may be many times the size of the eventual proof: while neural networks are in principle capable of processing large amounts of data, throughput suffers and scalability is a concern.
2. Data structures employed may be very opaque or "unnatural", containing artifice designed for efficiency rather than natural comprehension by a neural network.
3. Systems may be very sensitive to latency, which can result in the introduction of neural guidance systems crippling prover throughput and hence performance.

Attempting to solve these issues with a novel prover architecture, and exploring several options to improve overall efficiency, the prototype system LERNA[1] takes an alternative step toward useful neural automatic theorem provers.

2 Background

We assume basic familiarity with first-order logic, theorem proving, and neural networks [13].

2.1 Logic and Theorem Proving

First-Order Logic LERNA works with formulas in standard first-order logic with equality. Terms t and formulas ϕ are recursively defined as follows

$$\begin{aligned} t = {} & x \mid f\,(t_1, \ldots, t_n) \mid c \\ \phi = {} & \top \mid \bot \mid t_1 = t_2 \mid p\,(t_1, \ldots, t_n) \mid \phi_1 \Rightarrow \phi_2 \mid \phi_1 \equiv \phi_2 \mid \neg\phi \mid \\ & \phi_1 \wedge \ldots \wedge \phi_n \mid \phi_1 \vee \ldots \vee \phi_n \mid \forall x.\phi \mid \exists x.\phi \end{aligned}$$

where x is a variable, f is a n-ary function symbol, c is a constant, and p is a n-ary predicate symbol. Their meaning is defined as usual.

Automatic Theorem Provers An automated theorem prover (ATP) is a system able to automatically establish whether a formula (in first-order logic) is *satisfiable* or *unsatisfiable*; although, given the undecidability of this problem, ATPs may also return *unknown*. Both saturation-based provers (e.g. E [37], iProver [21], Vampire [33]) and SMT solvers (e.g. CVC4 [6] and Z3 [4]) utilise efficient proof calculi combined with highly-configurable search routines in order to explore a large search space efficiently.

2.2 Machine Learning and Theorem Proving

Despite the efficiency of modern ATP systems, they can still spend time exploring areas that a human mathematician would discard, and tuning such systems is, in general, extremely difficult [32]. This has led to the application of machine-learning techniques, with the eventual aim of an "intelligent" theorem prover able to learn from past experience to develop an intuition, discard uninteresting search space, and tune performance in a more principled way.

Previous work has focussed on premise selection [14,23,43,45], static strategy selection [3,25,24], dynamic (run-time) strategy selection [30] and more recently, direct proof guidance [44,16,17,15,26]. Proof guidance typically involves a form of machine-learned *heuristic* which biases proof search in some way, allowing the prover to avoid parts of the search space deemed uninteresting by the heuristic.

[1] **Le**arning to **R**eason with **N**eural **A**rchitectures. Lerna is also the lair of the mythical many-headed beast HYDRA. Source code available at `https://github.com/MichaelRawson/lerna`.

Work on integrating machine-learned heuristics into automatic theorem provers has relied on hand-engineered features [44,16,17] or other embedding methods [15,20], which have the advantage of simplicity and relative efficiency, but do not fully encode the syntactic structure of proof state and therefore lose information. By contrast, a neural method which takes into account all information (as utilised in this work) should allow for greater precision in proof guidance systems. Deep Network Guided Proof Search (DNGPS) [26] is an example of previous work in this area, which integrated a deep neural guidance system into the saturation-based prover E [37]. DNGPS achieved successful results, but suffered from the latency introduced into the system by the neural heuristic: despite processing only a reduced amount of the available proof state, the reduction in throughput necessitated a two-phase approach in which the prover was neurally-guided in the first phase, before falling back to traditional proof search in the second.

rlCoP The rlCoP system [17] is a connection-based reinforcement-learning prover which is not presently neurally-guided, but takes a similar approach to that taken in this paper and achieves impressive results.

Neural Networks for Formulae Neural networks are well-known tools for supervised learning [13], and combined with trainable convolution/pooling operators are suitable for processing large-scale data such as images [22].

Processing structured data such as logical formulae is a relatively new domain for neural networks. Some work attempts to use unstructured representations of such formulae, such as text, or build entirely-new models for a specific logic [7], whereas others attempt to re-use neural techniques for generic structures such as trees [2]. A promising direction in this area is recent research on neural methods working with graphs [5,19,36], which have already been applied to premise selection [45]. Graph neural networks tend to include network layers inspired by convolution operators in image-processing networks, combining information from neighbouring nodes (pixels) [19].

The MPTP Problem Set For training and evaluation purposes a set of valid propositions exported from the Mizar Mathematical Library [12] by the MPTP [42] system are used. Urban et al. [17] took a subset[2] of the large *M40k* problem set (containing 32,524 problems) and called it *M2k* (containing 2004 problems).

3 Design

In order to achieve the goal of a neural theorem prover without the disadvantages associated with neural approaches, a new design of theorem prover is required. Popular calculi used in existing ATPs tend to be unfriendly to neural guidance. For such a system, we desire the following from the calculus:

[2] `https://github.com/JUrban/deepmath/blob/master/M2k_list`

1. *Proof state must be reasonably-sized.* Attempting to evaluate large proof states structurally requires a lot of computation and resources. Saturation-based provers can have very large proof states, for example.
2. *Evaluation of states must be possible in parallel.* Machine-learning algorithms operate more efficiently in batches. Tree-based approaches (tableau *etc.*) lend themselves to this, whereas saturation provers are inherently sequential.
3. *Subgoals must be independent and self-contained.* If the prover has a notion of (sub-)goals which must be dispatched (such as in tableau provers), these should be independent of the rest of the search space, without e.g. unifiers. Otherwise, the learning system is trying to learn while blind to the context of the search.
4. *Subgoals must be intelligible.* Adding "noise" such as clausification obscures the original intuition behind a goal, at least for human observers. While this is not necessarily the case for machine-learning algorithms, it seems likely that removing structure and adding artefacts will reduce model performance.

We therefore implement a refutation prover based on a first-order tableaux calculus without unification, on non-clausal formulae. Each goal in this case is the set of formulae present on the tableau branch. In this context, proof state is small (only the current branch), evaluation of states is possible in parallel, each branch is independent and contains all information required, and all available structure from the original problem is kept.

3.1 Search

In the calculus (see Section 4) for this prover, there are two branching factors: each goal has a set of possible inferences, and each inference contains a set of possible sub-goals. To prove a goal, at least one inference must be proved. To prove an inference, all the inferences' sub-goals must be proved (e.g. shown to be unsatisfiable). A simple optimisation is that sub-goals may be shared between inferences, so search becomes a directed acyclic graph, alternating between goals and inferences (illustrated in Fig. 1).

Now the search graph can be explored: in each step, a leaf (goal) node is selected for expansion, and all resulting inferences and sub-goals are added to the graph. If a goal has no possible inferences, it is satisfiable and can be removed from the search space. On the other hand, if a goal is trivial (i.e. contains a contradiction), it is unsatisfiable and can be marked as proven. This idea is lifted to inferences: if an inference contains any satisfiable sub-goal, it too is satisfiable, whereas if an inference contains all unsatisfiable goals, it is unsatisfiable. Proof search continues until the timeout is reached

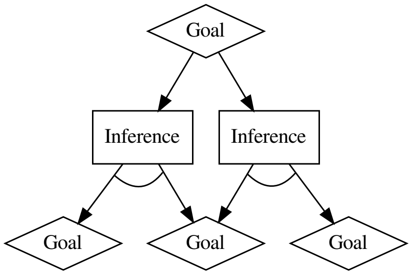


Fig. 1: Search in the LERNA system, showing shared sub-goals.

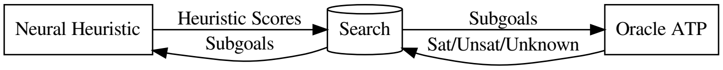


Fig. 2: Illustrating information-flow in the system.

or the root goal is shown to be (un-)satisfiable. In order to dispatch trivial subgoals quickly, an existing fast *oracle* ATP is used (see Section 5). This may mark goals as (un-)satisfiable, at which point no further exploration is required.

Search is biased by heuristic evaluation. The neural heuristic function (see Section 6) evaluates each goal and assigns a score corresponding to whether the network believes that the goal is satisfiable or unsatisfiable. In order to balance exploitation of promising directions and exploration of all parts of the search space, a principled UCT-based search algorithm is used, as in MonteCoP [8]. At each sub-goal g, the prover chooses the inference i with subgoals s according to

$$\max_{i \in g} \left[\underbrace{\min_{s \in i} (\mathsf{score}(s))}_{\text{exploitation}} + c \times \underbrace{\sqrt{\frac{\ln \mathsf{visits}\,(g)}{\mathsf{visits}\,(i)}}}_{\text{exploration}} \right]$$

where score gives the heuristic score, visits gives the total number of visits to that node so far, and c is the exploration parameter (theoretically $\sqrt{2}$). The subgoal with the minimal score is then selected: this prioritises subgoals considered possibly satisfiable by the heuristic, as satisfiable subgoals allow large parts of the search space to be pruned.

3.2 Architecture and Prototype Implementation

The system aims to consume all available CPU and GPU resources as efficiently as possible. To that end, proof search is asynchronous: the search algorithm generates new sub-goals, which are placed on two separate queues: one for the oracle ATP, another for heuristic evaluation. Proof search then continues elsewhere, while the oracle ATP is called in parallel on each sub-goal (consuming all available CPU) while the heuristic consumes batches of subgoals, efficiently utilising the available computational resource. As information flows backwards from these processes, the search process updates its information about a given sub-goal and propagates that information upwards to the sub-goal's parent inferences, to influence future proof search: see Fig. 2.

The prototype implementation (minus the heuristic) is currently just under 3,000 lines of Rust, not including the TPTP format parser or the implementation of perfect sharing. Python 3 was used for the heuristic due to the large number of libraries available for neural network implementation in Python. The heuristic is implemented as a server, communicating with the main prover via a TCP socket. In principle this allows for the heuristic to be a shared resource with a centralised heuristic server, or a load-balanced cluster.

$$\textsc{contradiction}\;\frac{}{\phi, \neg\phi, \Gamma} \qquad \textsc{equal}\;\frac{t = s, \phi\left[t/s\right], \Gamma}{t = s, \phi, \Gamma} \qquad \textsc{implies}\;\frac{\neg\phi, \psi, \Gamma}{\phi \Rightarrow \psi, \Gamma} \qquad \textsc{equivalent}\;\frac{\neg\phi, \neg\psi, \Gamma \qquad \phi, \psi, \Gamma}{\phi \equiv \psi, \Gamma}$$

$$\textsc{conjunction}\;\frac{\phi_1, \phi_2, \ldots, \phi_n, \Gamma}{\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n, \Gamma} \qquad \textsc{disjunction}\;\frac{\phi_1, \Gamma \qquad \phi_2, \Gamma \qquad \ldots \qquad \phi_n, \Gamma}{\phi_1 \vee \phi_2 \vee \ldots \vee \phi_n, \Gamma}$$

$$\textsc{instantiation}\;\frac{\forall x_1, x_2, \ldots x_n.\phi[f(x_1, x_2, \ldots x_n)/x], \forall x.\phi, \Gamma}{\forall x.\phi, \Gamma} \qquad \textsc{non-empty}\;\frac{\phi[k/x], \Gamma}{\forall x.\phi, \Gamma} \qquad \textsc{exists}\;\frac{\phi[k/x], \Gamma}{\exists x.\phi, \Gamma}$$

Fig. 3: A complete inference system for Lerna. Rules for negation are as usual and not shown here for brevity. In rule instantiation, f is a function symbol of arity n in the conclusion's signature and $x_1 \ldots x_n$ are fresh for the conclusion. In rules non-empty and exists, k is fresh for the conclusion. $\phi[t/s]$ is a capture-avoiding substitution replacing t for s in ϕ.

4 Calculus

The proof calculus used in the above architecture may be extremely general: in fact, any function from goals to a finite set of possible inferences (themselves finite sets of sub-goals) will suffice, as long as each goal remains independent of any other such that the heuristic function can process all available information. If the inference system is complete, there are no additional constraints such as orderings or fairness to ensure the completeness of the prover, as the balanced search algorithm (see Section 3) will ensure this.

Lerna presently implements a refutation tableaux calculus [35] without unification. The calculus described is deliberately naïve in order to easily satisfy the design constraints given above, but may be replaced by a stronger calculus in the future. A naïve calculus is not necessarily a problem as the heuristic should select promising areas to explore and ignore uninteresting sub-goals. However, a more efficient calculus would improve performance where the heuristic fails.

Refutation Tableaux In order to show a conjecture C from a set of axioms A_i, it suffices to negate C and then show that the resulting conjunction $A_1 \wedge A_2 \wedge \ldots \wedge \neg C$ is unsatisfiable. A set of inference rules of the form

$$\frac{\Gamma_1 \qquad \Gamma_2 \qquad \ldots \qquad \Gamma_n}{\Delta}$$

where Γ_i, Δ are sets of formulae and $\neg\left(\Gamma_1 \wedge \Gamma_2 \wedge \ldots \Gamma_n\right) \Rightarrow \neg\Delta$ is an unconditional tautology, form a refutation calculus. Proofs in this calculus can be expressed by closed trees of inference rules.

$$\overset{\textsc{double-neg}}{\frac{\phi, \Gamma}{\neg\neg\phi, \Gamma}} \qquad \overset{\textsc{conj-assoc}}{\frac{\phi \wedge \psi \wedge \pi, \Gamma}{\phi \wedge (\psi \wedge \pi)}} \qquad \overset{\textsc{disj-prop}}{\frac{\phi, \Gamma}{\phi \vee \bot, \Gamma}} \qquad \overset{\textsc{refl}}{\frac{\top, \Gamma}{t = t, \Gamma}} \qquad \overset{\textsc{free}}{\frac{\phi, \Gamma}{\forall x.\phi, \Gamma}}$$

Fig. 4: Some simplification rules implemented in Lerna. In rule free, x is free in ϕ. Several other rules are implemented.

Complete Inferences The inference rules in Fig. 3 form a complete inference system, by analogy with a first-order tableaux calculus without unification. A difference and point of interest is the rule for instantiating universal quantifiers: instead of instantiating a variable with any possible term t — an infinite space — it is instantiated with one function symbol (or constant) at a time, quantifying over new variables as needed. This allows for instantiating any term over multiple inference rules (effectively enumerating the Herbrand universe for the goal), but without an infinite number of possible inferences at any point. Equality is handled by a rule rewriting classes of equal ground terms. Both of these rules are complete yet inefficient, but both are likely to be used only a few times in order to provide enough of a "hint" to the oracle system for it to find a proof.

Weakening A weakening rule is an important part of Lerna's calculus, since the instantiation and equal rule can produce a large number of formulae, some of which must be removed to help the oracle to prove the goal. Each application of the rule removes some amount of information from the goal in order to simplify it — this is sound and corresponds to removing an axiom from proof search. The rule is merely

$$\overset{\textsc{weaken}}{\frac{\Gamma}{\phi, \Gamma}}$$

Simplifications Before each inferred goal is added to proof search, it is simplified, removing tedious inferences such as double-negation elimination and generally reducing the search space. Fig. 4 gives example simplification rules.

5 Oracle

One problem with the calculus as described is that proofs can be quite lengthy, even if the goal is relatively trivial. To rectify the situation, new goals generated by ongoing proof search are enqueued for attempted proof by an existing *oracle* ATP system, as described in Section 3. In our prototype implementation we use the mature Z3 SMT solver [4], which supports quantified first-order logic via a combination of decision procedures for decidable fragments (such as the Bernays-Schönfinkel class of formulae), and heuristic quantifier instantiation routines [11]. Z3 is attractive for this application due to its low startup times and its ability to produce both satisfiable and unsatisfiable results.

Lerna uses Z3 as an external system (it could be replaced by an alternative ATP), running it with its Model-Based Quantifier Instantiation heuristic for 20 milliseconds. This was chosen as the shortest time in which the oracle can dispatch a reasonable amount of trivial goals (and in fact Z3 is so strong it dispatches some goals immediately: see Section 7). Longer oracle runtimes might produce better performance in future, but for this work longer runtimes begin to conflate the performance of the oracle and the performance of the system as a whole. This application is unusual for ATP systems — very short runtimes, and a mix of true and false problem statements.

Acting as a Preprocessor Lerna might also be seen as an intelligent preprocessor for existing ATPs in this setting: existing theorem provers are known to be sensitive to small changes in their input [40], and generally make little attempt to split their input into smaller sub-goals, for parallelism [41] or otherwise. The system can therefore act as an adaptor for any existing ATP, adding parallelism opportunities and "smoothing out" sensitivity to input syntax.

6 Learned Heuristic

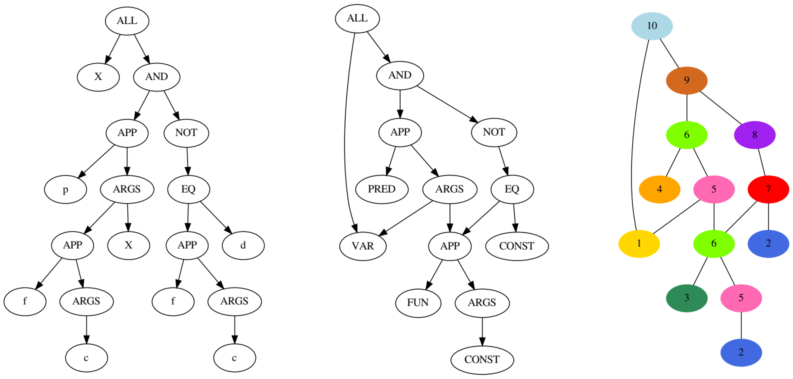
A suitable heuristic function for the system must predict a value between 0 and 1 for a given formula F, where 0 represents a satisfiable goal and 1 represents unsatisfiability, based on a set of tagged formulae seen in previous proof search. Although the data is collected by running the system itself and might be considered *reinforcement* learning, for this approach data collection and learning were considered separately and hence forms a classic supervised-learning problem.

6.1 Data Collection

A large dataset of satisfiable and unsatisfiable goals were collected by running the unguided prover on the *M40k* dataset for 10 seconds. As soon as the prover determines the satisfiability of any sub-goal, the formula it represents and its status is recorded. This resulted in 18,340 unsatisfiable examples and 1,845,267 satisfiable examples, occupying 6GB of disk space. The dataset is very imbalanced (due to a combination of weakening rules producing a large number of trivially-satisfiable examples, and to immediate prover termination after the goal is shown to be unsatisfiable), at a ratio of around 100:1.

6.2 Translation to Graphs

Wang et al. [45] give a translation from higher-order formulae to directed graphs, and a similar scheme is used here. Constants, function symbols, predicate symbols, and bound variables are given their own node. Applications of functions and predicates to arguments are represented as an "application node" with two children: the symbol node and an "argument list" node representing the list



(a) AST.

(b) Nameless DAG.

(c) Final result.

Fig. 5: The translation process for $\forall x. \, [p\,(f(c), x) \wedge \neg\,(f(c) = d)]$ to a graph, as seen by the neural network.

of arguments. Propositional connectives and equality have the obvious representation, while quantifiers have two children: the variable they bind and their sub-formula.

To produce an input graph from a formula F, the formula is first parsed into an abstract syntax tree. Common sub-trees up to α-equivalence [1] are merged, then the resulting directed acyclic graph has any named-symbol nodes replaced with an opaque, nameless label such as "predicate" or "variable" — since distinct symbols remain as distinct nodes under this scheme, no information is lost other than the natural-language semantics of the symbol name. In practice, undirected graphs improved model performance so the graph is made undirected before encoding node labels as one-hot inputs to produce the final input graphs. An example formula's translation is shown in Fig. 5.

6.3 Augmentation

One possible solution [39] to the problem of classification on imbalanced domains is to synthesise new data for under-represented classes — in this case unsatisfiable formulae — from existing data by augmenting it. An example is augmenting image data by cropping, flipping or adding noise to existing images. There are many possible ways to augment formulae graphs. For this prototype, a simple approach is taken in which a small number of nonsense formulae are added to the graph by randomly adding nodes/edges where appropriate. This approach has the advantage of exposing the network to "noise" such as additional axioms which might well occur in practice, but if the network is adequately capable of filtering these then no new formulae are actually seen.

6.4 Neural Architecture

In a typical convolutional network architecture for images [22], there are a series of filtering stages, followed by a densely-connected neural network. Each filtering stage intuitively combines data from local features (via *convolution*), then reduces the dimensions of the image (via *pooling*) for the next stage. Graph neural networks have analogous convolution [19] (combining information from neighbouring nodes) and pooling [10] (merging nodes to reduce the size of the input graph) operators. A brief period of experimentation with these operators yielded the following network architecture, shown in Fig. 6.

1. Input. A graph G consisting of one-hot encoded nodes N and edges E.
2. Embedding. Each node is mapped to an embedding vector of size 64 via a trained dense embedding.
3. Initial Convolution. 4 convolution layers are applied to the graph with rectified linear activations. This yields a graph of the same size, but with information exchanged between nodes.
4. Convolution/Pooling. Similar convolution layers are then passed through *top-k* [10] layers, retaining $k = 60\%$ of the graph's nodes. This is repeated 3 times, reducing the size of the graph considerably.
5. Convolution/Max-Pooling. A final convolution layer feeds into a max-pooling layer, combining all remaining node data into one datum, and dropping the edge data.
6. Fully Connected. A fully-connected hidden layer with rectified linear activation halves the input size.
7. Fully Connected/Softmax. A fully-connected final layer outputs two class labels, with softmax activation.

It is not claimed that this is the optimal configuration, and no grid search has yet taken place to optimise the network architecture or hyper-parameters. To reduce over-fitting, dropout [38] is applied in convolutional and fully-connected layers, $p = 0.1$.

6.5 Implementation and Training

This architecture was implemented with the PyTorch [29] neural network library, combined with a graph-processing ("geometric") extension library, PyTorch Geometric [9], which together provide facilities for automatic differentiation, GPU-accelerated training, pre-programmed layers for graph processing, and various utilities. The dataset is split into a large training set and a smaller test set (200 balanced examples), since unsatisfiable examples were time-consuming to obtain in this setting. The unsatisfiable training data were then augmented as described in section 6.3 to produce a balanced total training set of 3.5 million examples. The network was trained on commodity desktop hardware with a mid-range GPU [3] for 8 epochs/24 hours, optimising a negative log-likelihood loss function.

[3] NVIDIA® GeForce® GT 730.

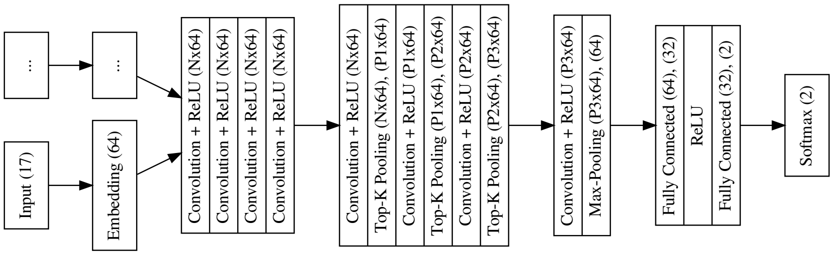
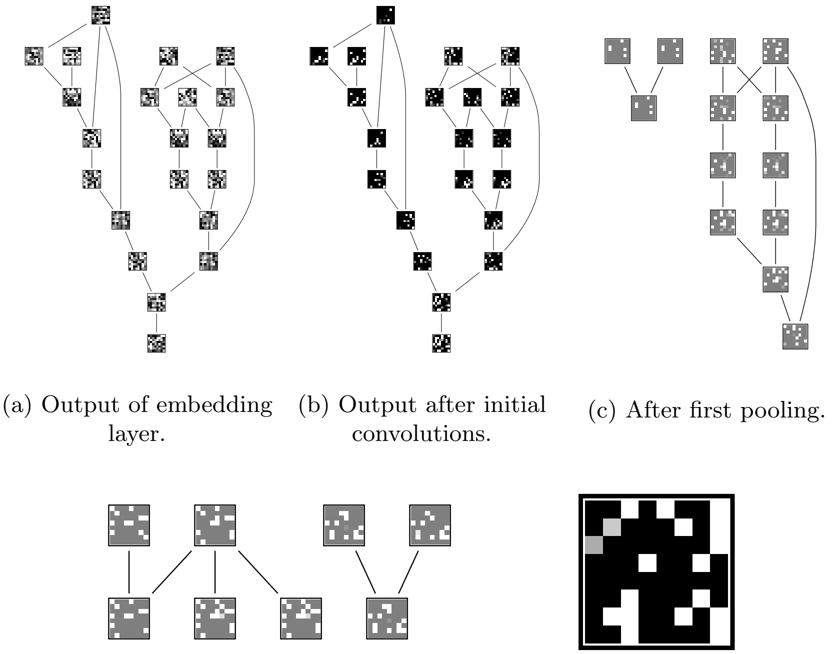

...
...
Input (17)
Embedding (64)
Convolution + ReLU (Nx64)
Convolution + ReLU (Nx64)
Convolution + ReLU (Nx64)
Convolution + ReLU (Nx64)
Convolution + ReLU (Nx64)
Top-K Pooling (Nx64), (P1x64)
Convolution + ReLU (P1x64)
Top-K Pooling (P1x64), (P2x64)
Convolution + ReLU (P2x64)
Top-K Pooling (P2x64), (P3x64)
Convolution + ReLU (P3x64)
Max-Pooling (P3x64), (64)
Fully Connected (64), (32)
ReLU
Fully Connected (32), (2)
Softmax (2)

Fig. 6: The neural network architecture. Initially there are N nodes, then after pooling there are P_1, P_2, P_3 nodes. Node-level embedding layers are shown per-node, graph-level convolutional and pooling layers are shown per-graph.



(a) Output of embedding layer.

(b) Output after initial convolutions.

(c) After first pooling.

(d) After second pooling.

(e) After max-pooling.

Fig. 7: Computation in the neural network, showing intermediate values involved in the network (correctly) predicting the satisfiability of an input formula.

Table 1: Accuracy metrics for the neural heuristic.

Metric	**Score**	**Metric**	**Score**
Accuracy	93.0%	True Positive	99
Precision	0.990	True Negative	87
Recall	0.884	False Positive	13
F_1	0.934	False Negative	1

Table 2: Total successful proof attempts on the *M2k* dataset.

Configuration	**Proofs**
Z3 (10s, as baseline)	1216
Z3 (20ms, as oracle)	711
Lerna, unguided (10s, with oracle)	969
Lerna, guided (10s, with oracle)	1023

6.6 Network Evaluation

The network was evaluated on the balanced test set of 200 examples, as described. Various metrics for accuracy are shown in Table 1. While these results are very promising, it should be emphasised that it is unclear how effective a train/test split is in this setting (since similar subgoals may occur in both sets, even with proper data hygiene), and that this network is not attempting to determine the satisfiability of arbitrary formulae, merely those that occur in proof attempts on the *M40k* dataset. The higher precision and lower recall values are likely an artefact of the augmentation process. However, even with these caveats, the network performance is surprising and is practically useful for improving proof search in this dataset.

7 Experimental Results

To show that neural guidance can improve the performance of Lerna the system was run with and without guidance for 10 seconds on all available CPU cores. All results were collected on commodity desktop hardware[4].

Table 2 shows the total number of theorems proved using various configurations of Z3 and Lerna on the *M2k* dataset. Z3 ran for a full 10 seconds to establish baseline performance, then as an oracle for 20 milliseconds to determine the number of “trivial” problems. Lerna ran on an identical dataset, first without guidance from the neural heuristic, then with guidance. With neural guidance Lerna was able to solve an additional 54 problems and overall Lerna was shown to be complementary to Z3, proving 114 problems that Z3 was unable to solve on its own, and 40 that neither unguided Lerna nor Z3 could solve. Conversely, Z3 was able to solve more problems in total, which is unsurprising given the maturity of the tool. These results show that Lerna is able to learn from experience and complement an existing ATP.

[4] Intel® Core™ i7-6700 CPU @ 3.40GHz, 16GB RAM.

8 Future Work

Given the prototype nature of this work, we have included a detailed discussion on future directions. As Lerna is a very new system, there is likely much to be gained by simple engineering and tuning: for example, the UCT exploration parameter c has been left at its theoretical optimum value $\sqrt{2}$, but it is likely that a higher value will account for neural network inaccuracies and hence improve performance. Training on, benchmarking with, and optimising for other datasets (such TPTP or SMT-LIB) is also left as future work.

Proof Search Lerna is well-suited for long-term proof search attempts in mathematics, such as those employed in the AIM project [18]: search is stable over time and does not produce a combinatorial explosion in the same way that some traditional systems tend to after a short period. Additionally, the amount of information ("confidence") in the system grows over time, as a result of a growing number of oracle invocations and neural network evaluations. Proof search can in principle be manually inspected more easily than in saturation-based provers to examine promising subgoals and remove known falsehoods from the search space. The authors hope to explore applying the system to this interesting domain.

Another future direction for proof search is a principled incomplete mode where branches deemed sufficiently uninteresting by the heuristic are pruned, perhaps in response to resource constraints as in limited resource strategies [34]. This approach, while clearly incomplete, would significantly accelerate proof search in the direction of more promising search within the available resources.

Prover Calculus The calculus currently employed is deliberately naïve and extensions should be explored. In particular, the simplification routines can be improved to remove more trivial sub-formulae as, while in general the oracles' preprocessing will remove these, they serve as noise for the neural network and might also increase the number of inference steps required to reach a proof. As one possible view of this approach is as an intelligent preprocessor for an existing ATP, more aggressive and/or weakening inferences might be included in the calculus. For instance, *prenexing* (or conversely *miniscoping*) formulae can have a significant effect on proof search for some theorem provers, so including suitable quantifier-manipulation rules might prove to be a useful extension.

Ideas from other refutation-tableaux calculi could well be suitable for this system. The authors are attempting to integrate an adapted connection rule from the non-clausal connection calculus [27], as used in nanoCoP [28], in order to reduce the number of proof steps required to instantiate universal quantifiers. Finally, this prover architecture can support other logics without excessive modification. Given that Z3 is already capable of supporting many *theories*, such as arithmetic or datatypes, a many-sorted first order logic such as those described by SMTLIB or the TFF0 dialect of TPTP seems appropriate.

Oracle While Z3 is a strong theorem prover in its own right and performs well here, it remains to be seen if it is the best for this application. Other ATPs (or counter-example-finding systems) should be explored. A *portfolio* of several oracle systems working in tandem might also be considered, although of course this will eventually retard proof search linearly in the number of systems present. Reducing the number of oracle invocations is another area for optimisation. Currently, the system calls an oracle for every new sub-goal generated. It seems unlikely that the sub-goal is materially easier to dispatch than its parent (especially in the case of propositional inferences that do not split the goal), so heuristically or probabilistically removing such subgoals from the oracle's queue is a possible area for improvement. Lerna does not currently use any information from the oracle beyond its status: using auxiliary information such as satisfying models or unused formulae could well aid proof search.

Machine-Learned Heuristic Many other graph-based neural architectures are possible. PyTorch Geometric alone currently includes nearly 40 other graph-specific neural layers pre-programmed from the literature[5]. Neural models specifically for theorem proving are relatively under-studied. To combat this, data used for this paper will be published in the near future so that the machine-learning community can improve upon our simple models. Different approaches to formula-to-graph translation, symbol embeddings, data augmentation, and model integration may also be explored.

9 Conclusions

The introduced prototype Lerna system successfully implements a theorem prover with a neural heuristic processing the entire proof state, structured as a graph. After training on data automatically generated by the prover system, the neural network approach is shown to be practically useful for improving proof search performance. A number of approaches (batching, oracle invocations, parallelism) are employed to improve system efficiency. While the prototype is not yet a successful state-of-the-art ATP, it has some unique desirable properties, among them simplicity, parallelism, parametricity with respect to calculus/oracle/heuristic, and introspection of proof state. The general approach is flexible and presently unexplored.

Acknowledgements The authors wish to thank Josef Urban and his group in ČVUT, Prague for their help and encouragement with early iterations of this work, and for supplying the Mizar dataset used in this paper.

References

1. Barendregt, H.P., et al.: The lambda calculus. North-Holland Amsterdam (1984)
2. Bowman, S.R., Potts, C., Manning, C.D.: Recursive neural networks can learn logical semantics. arXiv preprint arXiv:1406.1827 (2014)

[5] `https://rusty1s.github.io/pytorch_geometric/build/html/modules/nn.html`

3. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine learning for first-order theorem proving. Journal of automated reasoning **53**(2), 141–172 (2014)
4. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems. pp. 3844–3852 (2016)
6. Deters, M., Reynolds, A., King, T., Barrett, C.W., Tinelli, C.: A tour of CVC4: how it works, and how to use it. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. p. 7 (2014)
7. Evans, R., Saxton, D., Amos, D., Kohli, P., Grefenstette, E.: Can neural networks understand logical entailment? arXiv preprint arXiv:1802.08535 (2018)
8. Färber, M., Kaliszyk, C., Urban, J.: Monte carlo connection prover. arXiv preprint arXiv:1611.05990 (2016)
9. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
10. Gao, H., Ji, S.: Graph U-Net (2018), preprint at `https://openreview.net/forum?id=HJePRoAct7`
11. Ge, Y., De Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: International Conference on Computer Aided Verification. pp. 306–320. Springer (2009)
12. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Mizar in a nutshell. Journal of Formalized Reasoning **3**(2), 153–245 (2010)
13. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall PTR (1994)
14. Irving, G., Szegedy, C., Alemi, A.A., Een, N., Chollet, F., Urban, J.: DeepMath — deep sequence models for premise selection. In: Advances in Neural Information Processing Systems. pp. 2235–2243 (2016)
15. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: International Conference on Intelligent Computer Mathematics. pp. 292–302. Springer (2017)
16. Kaliszyk, C., Urban, J.: FEMaLeCoP: Fairly efficient machine learning connection prover. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 88–96. Springer (2015)
17. Kaliszyk, C., Urban, J., Michalewski, H., Olšák, M.: Reinforcement learning of theorem proving. In: Advances in Neural Information Processing Systems. pp. 8822–8833 (2018)
18. Kinyon, M., Veroff, R., Vojtěchovskỳ, P.: Loops with abelian inner mapping groups: An application of automated deduction. In: Automated Reasoning and Mathematics, pp. 151–164. Springer (2013)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Komendantskaya, E., Heras, J.: Proof mining with dependent types. In: International Conference on Intelligent Computer Mathematics. pp. 303–318. Springer (2017)
21. Korovin, K.: iProver – an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008). Lecture Notes in Computer Science, vol. 5195, pp. 292–298. Springer (2008)

22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
23. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: MaSh: machine learning for sledgehammer. In: International Conference on Interactive Theorem Proving. pp. 35–50. Springer (2013)
24. Kühlwein, D., Schulz, S., Urban, J.: E-MaLeS 1.1. In: International Conference on Automated Deduction. pp. 407–413. Springer (2013)
25. Kühlwein, D., Urban, J.: MaLeS: A framework for automatic tuning of automated theorem provers. Journal of Automated Reasoning **55**(2), 91–116 (2015)
26. Loos, S., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. arXiv preprint arXiv:1701.06972 (2017)
27. Otten, J.: A non-clausal connection calculus. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 226–241. Springer (2011)
28. Otten, J.: nanoCoP: A non-clausal connection prover. In: International Joint Conference on Automated Reasoning. pp. 302–312. Springer (2016)
29. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch (2017)
30. Rawson, M., Reger, G.: Dynamic strategy priority: Empower the strong and abandon the weak. AITP 2018 (2018)
31. Rawson, M., Reger, G.: Towards an efficient architecture for intelligent theorem provers. AITP 2019 (2019)
32. Reger, G., Suda, M., Voronkov, A.: The challenges of evaluating a new feature in Vampire. In: Vampire Workshop. pp. 70–74 (2014)
33. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI communications **15**(2, 3), 91–110 (2002)
34. Riazanov, A., Voronkov, A.: Limited resource strategy in resolution theorem proving. Journal of Symbolic Computation **36**(1-2), 101–115 (2003)
35. Robinson, A.J., Voronkov, A.: Handbook of automated reasoning, vol. 1. Gulf Professional Publishing (2001)
36. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European Semantic Web Conference. pp. 593–607. Springer (2018)
37. Schulz, S.: E–a brainiac theorem prover. Ai Communications **15**(2, 3), 111–126 (2002)
38. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
39. Sun, Y., Wong, A.K., Kamel, M.S.: Classification of imbalanced data: A review. International Journal of Pattern Recognition and Artificial Intelligence **23**(04), 687–719 (2009)
40. Sutcliffe, G., Melville, S.: The practice of clausification in automatic theorem proving (1996)
41. Suttner, C.B., Schumann, J.: Parallel automated theorem proving. In: Machine Intelligence and Pattern Recognition, vol. 14, pp. 209–257. Elsevier (1994)
42. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. Journal of Automated Reasoning **37**(1-2), 21–43 (2006)
43. Urban, J.: MaLARea: a metasystem for automated reasoning in large theories. ESARLT **257** (2007)

44. Urban, J., Vyskočil, J., Štěpánek, P.: MaLeCoP machine learning connection prover. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 263–277. Springer (2011)
45. Wang, M., Tang, Y., Wang, J., Deng, J.: Premise selection for theorem proving by deep graph embedding. In: Advances in Neural Information Processing Systems. pp. 2786–2796 (2017)