Using Neural Networks to Control Glut in the Active Logic Machine

Justin D. Brody^{1,2}, Bobby Austin¹, Omar Khater¹, Christopher Maxey³, Matthew D. Goldberg³, Timothy Clausner², Darsana Josyula⁴ and Donald Perlis²

¹Franklin and Marshall College, PO Box 3003, Lancaster PA 17604

²University of Maryland Institute for Advanced Computer Studies, College Park MD 20740

³University of Maryland Department of Computer Science, College Park MD 20740

⁴Bowie State University Department of Computer Science, 14000 Jericho Park Road, Bowie MD 20715

1. Introduction

In this paper, we address the problem of *inferential glut* – specifically the problem that symbolic reasoners tend to produce vastly more deductive information than they can tractably process. As a concrete example, let us imagine an agent which is searching for a specific book. Entering a study, it encounters, among other things, a bookshelf, a lamp, and a table. If the agent has a knowledge base with a large number of facts about lamps and tables, it may spend an inordinate amount of resources reasoning about lamps and tables, and such irrelevant reasoning can prevent it from drawing conclusions that are helpful for finding the book. In fact, such swamping can also arise immediately and devastatingly even without distractor entities, as discussed below. Of course, the agent could also have special information guiding the reasoning in ways appropriate to the specific task (e.g., books tend to be on shelves). But how is this special information obtained? Here we address this as a learning issue and report on preliminary results from our approach.

As an example, consider a round library with one continuous circular shelf along its circumference, where each book B has exactly one "left" neighbor left(B) and another "right" neighbor right(B), where these function symbols simply refer to those two neighbors of B. This might be encoded by the following rules K for some book a:

1. book(a)2. $book(X) \rightarrow book(left(X))$ 3. $book(X) \rightarrow book(right(X))$

Figure 1: The Knowledge Base K

NeSy'20/21: Workshop on Neuro-Symbolic Learning and Reasoning, October 25–27, 2021 (Virtual) ☆ jbrody2@fandm.edu (J. D. Brody)

^{© 02021} Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

We will consider an agent using the Active Logic reasoning engine ALMA [1], [2], [3], [4]. Active logic is a timestep logic in which reasoning proceeds by applying available inference rules to the contents of the knowledge base at the current timestep, which then produces a revised knowledge base at that next timestep. It does not apply "deductive closure" to derive consequences of consequences of sentences in the knowledge base, and there is no "final step" at which deductive closure is achieved. Reasoning continues in real time, with the agent revising its knowledge base as it goes. The intuition is that active logic reflects the reasoning of human agents, whose deductions take time and who can reason about the time a deductive process is taking.

For example, considering the theory K above, applying *modus ponens* allows ALMA to derive the sentences book(left(a)) and book(right(a)); these will thus enter the knowledge base at the next timestep. On the other hand, book(right(right(a))) requires two applications of *modus ponens* to derive, so it will not enter the knowledge base in the next timestep (but will in two timesteps). The ALMA reasoning engine supports the three basic kinds of inference found in [5]: resolution, forward inference and backward inference.

This limitation on the number of new derivations in a timestep has a number of advantages: one is that it avoids the *logical omniscience problem* [6], [1]; another is that it constrains the growth-rate of the knowledge base. However, it turns out that in the simple example given by K above, reasoning with these rules leads to enormous growth in inferences, far beyond the capacity of any current computer. In fact in n steps of reasoning the knowledge base above empirical evidence indicates that the reasoner gains $\Theta(2^{2^n})$ new inferences. The vast majority of these inferences will not be useful for any one particular reasoning task, and inferential glut is thus a serious issue even for active logic. In this paper, we propose an architecture for taming such glut based on using a neural network to assign priorities to potential inferences; we will present some preliminary results for this architecture.

Our main contributions are:

- 1. We develop an architecture for controlling glut.
- 2. We demonstrate the effectiveness of our architecture in a supervised learning task (learning when potential inferences will unify).
- 3. We demonstrate the effectiveness of our architecture in a simple reinforcement learning setting.

While our focus in this paper is on abstract reasoning, our ultimate aim is to apply our techniques to embodied agents reasoning in an environment. This changes the nature of the glut control problem in some immediate ways. For example, the derivations a situated agent needs are *prima facie* much shorter than those needed by an agent attempting to prove mathematical theorems. Also, the theories we work with are incomplete in that a number of the sentences that enter the knowledge base will come from observation rather than deduction. Thus the relevance of any particular premise will be at least partially determined by the environment rather than being implicitly present in the structure of the knowledge base at any given time. This impacts what kinds of models a network needs to learn – pure logical structure in the case of abstract reasoning versus logical structure in the context of a specific environment in our envisioned use case.

2. Related Work

Wang et al [7] represented logical sentences with syntax trees and fed the corresponding graphs into deep neural networks to perform *premise selection* – i.e. pick which available sentences in the knowledge base would likely be useful for reasoning toward the proof of a given conjecture.

Our work differs in that we select potential inferences rather than premises, and rather than reasoning toward a particular goal we reason according to a more generally defined heuristic function. In particular, this need not be goal-driven (in the sense of proving a given theorem); for example in section 5 we describe the training of a network heuristic which is aimed solely at ensuring that potential inferences can lead to new knowledge.

In addition to [7], a number of other works have used deep learning methods to guide reasoning: for example [8], [9] and [10]. With the notable exception of [8], most of these work in the context of automatic theorem proving. Our target is quite different – reasoning with an incomplete theory in an environment without an explicit sentence as a reasoning goal.

3. Glut

We divide glut into two varieties – *pre-inferential* and *inferential*. These arise from different sources and, in this preliminary work, we take different approaches to addressing them. The fundamental difference in this work is in how the models are trained – the basic architecture will be the same in both cases.

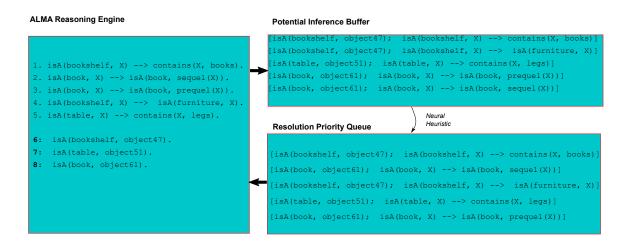
Pre-inferential glut arises from the way in which our reasoning engine works – each time a sentence is added to the knowledge base a very simple pattern match is performed to determine whether it can combine with any sentence to deduce new knowledge via an inference rule. Positive matches result in a *potential inference* – a pair of sentences which the system prepares to combine via an inference rule. Importantly, the simplicity of the pattern match can result in false positives. For example, if an ALMA knowledge base contains the rule $f(X, 0) \rightarrow g(X)$ and the sentence f(a, 1) is added to the knowledge base, then the pair $(f(X, 0) \rightarrow g(X), f(a, 1))$ will be taken as a potential inference even though f(a, 1) cannot be unified with f(X, 0) because of the conflicting second argument. We consider such potential inferences which will not unify to produce new knowledge to be *pre-inferential glut*. Because we can algorithmically check whether a potential inference will lead to a new deduction, training our architecture to filter out pre-inferential glut can be done with supervised learning. While filtering out pre-inferential glut could be achieved simply by attempting to apply resolution to each pre-inference, using a neural networks allows us to detect pre-inferential glut in constant time. Further, since we will ultimately apply a neural heuristic to every potential inference anyway, we are essentially able to do this filtering with no additional cost.

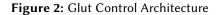
The second kind of glut we consider is *inferential glut* – these are potential inferences which will combine to produce new knowledge which is not relevant for the task at hand. Revisiting our original example of the agent searching for a book in a living room, we note that the agent's goal will not be well served by a long chain of reasoning about the lamp in the living room. The associated utility for such potential inferences is inherently tied to the task and environment in which the agent operates; thus training our network to diminish inferential glut seems best

suited to reinforcement learning.

4. Architecture

Figure 2 below shows an outline of our system's operation. Let us imagine that the knowledge engine begins with sentences 1 through 5 upon initialization and then adds sentences 6, 7 and 8. When adding these sentences, the system determines the potential inferences ¹ as described above and stores each such pair in a buffer. Each potential inference is then given a vector representation which is fed into one of a number of neural networks that define a heuristic function. This function assigns a priority p ($0) to each potential inference; the potential inference is then placed on a priority queue accordingly provided that <math>p < \theta$ for some threshold θ (note that lower p indicates a more immediate priority). Then, for some T > 0, the engine processes the top T elements of the priority queue² and attempts to apply the appropriate inference rule.





It is important to note that our system uses an **atomic step** which differs from the step of classical ALMA. Specifically, an atomic step is a *single* application of an inference rule to a *single* potential inference (with the exception of time-update rules, which are always applied). In classical ALMA a step consists of a single application of available inference rules to *all* potential inferences.

Since our system's notion of a step differs from that of classical ALMA, some remarks on how ALMA's theoretical properties differ are in order. First, we note that because ALMA is designed

¹At present, the system only works with applications of the resolution inference rule. Applications of forward and backward inference will be treated in future work.

²In a sense, this makes the inference trajectory in our system follows a kind of A* search, albeit without the proven guarantees of consistency on the heuristic function.

to operate as a time-sensitive reasoner [2] there are some inferences which cannot be deferred to a later time. In particular, ALMA has a built in notion of time at step t of reasoning Now(t) is in the knowledge base. This is updated by the rule Now(t) -> Now(t + 1) and it is crucial that such updates occur at each timestep. We thus assign such rules a priority of 0 and ensure that priority 0 potential inferences are applied independently of the process described above.

The soundness and completeness of classical ALMA have been discussed elsewhere [2], [3]; here we note that the our system shares ALMA's soundness (in the sense that if the knowledge base is consistent, then any application of an inference rule will preserve that consistency)³. Because ALMA does not, in general, apply every available inference at every step, reasoning will **not** usually be complete. It is worth noting, however, that for $\theta = 1$ and $T = \infty$ (i.e. using the entire priority queue at each step), our system reduces to classical ALMA and will thus inherit ALMA's completeness [3].

Thus our system provides a set of parameters that tune a tradeoff between completeness and efficiency. As we will see in the next section, the efficiency gains are both in terms of time and space: our control of the inference process can both reduce the number of sentences that must be stored and allow that system to draw valuable conclusions more quickly.

4.1. Neural Heuristic

While the training of the neural network that defines our heuristic function will vary in the two examples, discussed below, the vectorization algorithm and network architecture will remain constant.

Vectorization in this context refers to the process of turning the pairs of sentences in a potential inference into vectors which can become the input to a neural network. Ideally, such a transformation will balance preserving critical information about the structure of the sentences in the potential inference with representing such sentences concisely. In this preliminary work, we draw on the work of Wang et al [7] who proposed representing such sentences using graph neural networks. While we do not currently use a full implementation of graph neural networks, we use a vectorization algorithm that is inspired by them and could be thought of as "graph neural networks without message passing". Future work will explore using full graph networks; amongst other things we expect that this will allow our representations to effectively scale up to larger knowledge bases with longer sentences.

Our particular vectorization algorithm works with a fixed vocabulary for each knowledge base. This will consist of the names of constants, functions and predicates that occur in the knowledge base. Each of these is assigned a numerical index to use in encoding. We also assign such indices to syntactical tags. For example, for our knowledge base *K* (Figure 1) we might define an indexing function i by i(book) = 0, i(a) = 1, i(right) = 2, i(left) = 3, i(if) = 4, i(and) = 5, i(or) = 6, i(func) = 7, i(var) = 8). Such an indexing scheme allows to decorate the node of a syntax tree with a one-hot feature representation vector. See Figure 3 for an example of representing the sentence if(book(X), book(right(X))). Each node of the syntax tree is decorated with a feature vector, shown in the left of the figure. Each feature

³Importantly, we do **not** assert that starting from a consistent initial knowledge base guarantees that the knowledge base will always be consistent. In particular, observations can contradict previously held assumptions. As a paraconsistent logic, ALMA is designed specifically to detect and correct such inconsistency.

vector is a 1-hot vector indicating, for the given node, which vocabulary words are present and what syntactical tokens that node instantiates. Each sentence with an *n*-node syntax tree is then represented by a list of n feature vectors along with an $n \times n$ incidence matrix for the structure of the syntax tree itself. In our initial work, both structures were 0-padded to be represented as a 50-node syntax tree with possibly empty nodes (nodes with no connections and feature vectors of all 0s). The tree and feature-vector list were then flattened into a single vector which represented the sentence. Pairs of such vectors, representing the sentences of a potential inference, were then concatenated to form the input to the neural network.

nID	a	book	left	right	if	and	or	func	var	
0	0	0	0	0	1	0	0	0	0	
1	0	1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	1	
3	0	1	0	0	0	0	0	0	0	4 right
4	0	0	0	0	0	0	0	1	0	
5	0	0	0	0	0	0	0	0	1	

Figure 3: Vectorization: Every sentence is represented via an annotated graph of it's syntax tree as shown to the right (node ids are written in blue to the left of each node). The graph structure is encoded as an incidence matrix while the relevant information about each each node in the tree is encoded by a 1-hot representation.

The network itself had two hidden layers, each with 8 nodes and tanh activations. The output layer has a sigmoidal activation to keep our priorities between 0 and 1 (having bounded priorities is helpful for analysis). The loss and training differ from scenario to scenario and will be described individually in the sections which follow.

5. Pre-Inferential Glut Control

To study the system's ability to control pre-inferential glut, we worked with an initial knowledge base having a single sentence of the form:

$$f(x_0, y_0, t) \land f(b, y_1, t) \to g(x_0, b, y_0, t)$$

along with 50 observations of the form

for i an integer with $0 \le i < 50$ and d_i random integers with $0 \le d_i < 100$. Logically, there are 100 conclusions that can be derived from these initial sentences (of the forms $g(a, b, d_i, i)$ and $g(b, b, d_i, i)$ for $0 \le i < 50$).

The potential inferences generated by this system include pairs $g(a, b, d^0, t^0), g(a, b, d^1, t^1)$; these will unify if, and only if, $d^0 = d^1$ and $t^0 = t^1$. This leads to an overall sparsity of potential inferences which will unify within the space of generated potential inferences, indicating a real need for glut control.

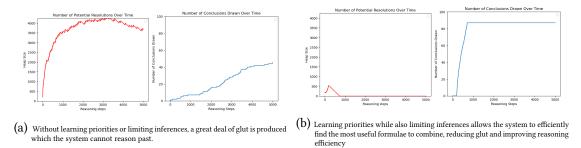


Figure 4: After training, our system is able to keep the size of the priority queue small and draw most of the inferences available to it.

During training, an epoch consisted of running the reasoner for 5000 steps (without priority control; i.e. $T = 1, \theta = 1.0$) and training on the potential inferences generated, with ground-truth labels generated algorithmically. Specifically, for each potential inference, we generate a label of 0 if the pair unifies and a label of 1 if the pair does not unify. Thus, for any potential inference p, the output of the network can be thought of as an estimate of $1 - v_p$, where v_p denotes the probability that p will unify. This has the effect of prioritizing potential inferences which are likely to unify.

The network is trained with mean-squared-error loss and the ADAM optimizer [11]. We trained the network until the accurracy was above 0.999 and the loss was below 1×10^{-5} .

To test the system, we ran with $T = 1, \theta = 0.7$. The results, shown in Figure 4, illustrate that after training of the heuristic network the system is able to limit the number of potential inferences that need to be considered *and* that the system is able to reason past the glut. In the first run (Figure 4a) the system used random priorities. In this situation, the number of potential inferences grows very rapidly and, since the reasoner is effectively only making random choices from the available potential inferences, is only able to combine less than 50 potential inferences to infer new sentences. On the other hand, after a modest amount of training (Figure 4b) the network is very easily able to derive over 90 of the available inferences and the number of potential inferences goes down very quickly.

6. Inferential Glut Control

Our ultimate aim is to control *inferential glut* – potential inferences which will lead to new information that is not useful for the task at hand. To control this, we set up the problem of glut control as a reinforcement learning problem. Any given scenario will have an associated reward function that is based on the particular sentences an agent has in its knowledge base.

Recall that for a trained heuristic function and given parameters θ , T, we have a well determined way of choosing which potential inferences will be applied at any given timestep. In reinforcement learning this corresponds to a *policy* π . Then we define our glut control problem as a reinforcement learning problem as follows

1. An *action a* corresponds to the selection of a particular potential inference.

- 2. The *state s* of the agent is determined by the set of sentences in the agent's knowledge base.
- The *value* (with respect to π) of any pair (s, a) is the expected return (total reward for executing the policy after selecting a) that the agent will receive from selecting premise a when the knowledge base is in state s if the policy π is followed for the entire episode. It is denoted by Q_π(s, a) or just Q(s, a) since our policy will always be that determined by the neural heuristic.

We approximate the function Q by training the neural heuristic network via reinforcement learning. In our case we specifically employed Q-learning to train a deep Q network to approximate the value using the standard techniques in [12]. To date, our network encodes only the action and not the state, thus we are currently learning an approximation of $Q(\cdot, a)$ rather than Q itself. In the situation we will examine, this still proves tremendously useful.

To test our setup, we worked with the knowledge base K from the introduction. While simple, these axioms seem to lead to a very rapid ($\Theta(2^{2^n})$) growth in the size of the knowledge base⁴; this is due to the fact that with resolution the system learns not only new literals (e.g. statements like book(left(left(right(a)))) but also general rules (such as book(X) -> book(left(left(right(X))))) and the maximal number of function applications involved in the latter doubles at each time step. Thus this simple axiom system already presents a real challenge for a glut-control system.

We trained our system using a network structure identical to that used in the pre-inferential scenario above. Our reward function assigned a reward equal to the total number of new instances of the function right in the knowledge base; e.g. a derivation of

book(right(left(right(a))))

would yield a reward of 2. We trained our network as a deep Q network [12] using epsilon greedy action selection with ϵ decaying linearly from 1 to 0.1 over the course of 300000 iterations and a discount factor of 0.99.

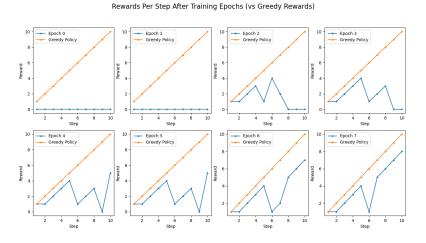
Some specifics of our training process are as follows. Following [12], we used *experience replay*. Saving sequences of (*action, reward*) pairs in a buffer we trained on random samples from the buffer every 100 iterations – this gives the network a better chance of training on random samples rather than on a highly correlated sequence of immediately succeeding states. Stability was also improved by applying *policy iteration*; we kept both a target model with parameters θ' and a current model with parameters θ ; the latter used the former to train as described below, with the target model being reset to the current model every 300 iterations. Each training episode consisted of a run of 10 atomic steps and we trained for 1000000 episodes. The loss function associated with any potential inference *a* was the standard one for *Q*-learning:

$$L(s, a) = (Q_{\theta}(a) - (r_{s,a} + Q_{\theta'}(a'))^2)$$

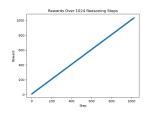
where *s* denotes the state of knowledge base at a given time, *a* denotes a particular action, $r_{s,a}$ denotes the reward associated with taking action *a* from state *s*, *s'* denotes the state of the

 $^{^{4}}$ The actual computer simulation used different predicate names, we use these here to connect with the theory K discussed in the introduction.

knowledge base that results from applying action a in state s and a' is the action with maximal $Q_{\theta'}$ -value from state s'. Essentially, L measures the degree to which the Q function satisfies the *Bellman equation* [12], [13], modified so that the estimate of future rewards uses the target network rather than the current network.



(a) Training for 10 Atomic Steps. As the system trains, the performance converges on the greedy strategy of increasing the number of rights in a token by one at each step



(b) Testing on 1024 Steps. Since the system only trained on 10 steps, the results show that system has learned the strategy of applying a single rule to obtain a new right at each step.

Figure 5: Training Results: The strategy learned for 10 steps extrapolates to a large number of steps.

Despite the massive growth discussed above, and despite being hobbled by only taking actions (and not states) as input, our system was able to learn an effective strategy for increasing rewards – namely by applying Rule 3 to the book with the most compositions of right in the knowledge base. For example, after 13 steps the system has learned that a composition of 13 rights with a is a book. At step 14 it applies 3 to what is has just learned to learn that a composition of 14 rights with a is a book.

Since our learning algorithm is stateless, the heuristic function cannot have any notion of which reasoning step it is working with. Under such a constraint, the policy learned must treat all time steps equally. Thus, since the policy learned is clearly the optimal policy for an agent with a single timestep, it must be the optimal policy which treats all timesteps equally⁵.

⁵For an agent that has exploitable information about how many time steps it will reason for, this greedy policy

It is worth emphasizing that learning the greedy strategy is a massive improvement over what would have been possible in classical ALMA. Whereas our system was able to efficiently reach a conclusion with 1024 compositions of the function right, because of the massive glut produced classical ALMA would not have been able to even reach 10 compositions. In fact, running the system for even 5 steps crashed our workstation with 384 gigabytes of memory. Thus even simple reasoning systems need some mechanism for glut control, and our system has the potential to effectively control glut without excessive sacrifices in reasoning power. In future work we will develop the system further and study it's effectiveness in more complex reasoning domains.

7. Conclusions and Future Work

We have presented neural network-based architectures to control both pre-inferential and inferential glut. While the studies described herein are preliminary in nature, they show promise for our approach in that we were able to effectively control glut in some simple scenarios. While these initial problems were simple, they already demonstrated significant levels of glut needing control. Future work will make the networks involved more robust (critically, we will represent state in our reinforcement-learning network) and explore the use of our architecture in increasingly complex environments.

8. Acknowledgements

Work by Brody, Maxey, Goldberg, Clausner, Josyula and Perlis was partially supported by DARPA grant DARPA-PA-19-03-01-01-FP-037 awarded to Perlis. Work by Austin and Khater was supported by a Hackman Summer Scholar award to Brody by Franklin and Marshall College.

References

- J. J. Elgot-Drapkin, D. Perlis, Reasoning situated in time i: Basic concepts, Journal of Experimental & Theoretical Artificial Intelligence 2 (1990) 75–98.
- [2] J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe, D. Perlis, Active logics: A unified formal approach to episodic reasoning, Technical Report, 1999.
- [3] M. L. Anderson, W. Gomaa, J. Grant, D. Perlis, Active logic semantics for a single agent in a static world, Artificial Intelligence 172 (2008) 1045–1063.
- [4] K. Purang, Alma/carne: implementation of a time-situated meta-reasoner, in: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001, IEEE, 2001, pp. 103–110.
- [5] S. Russell, P. Norvig, Artificial intelligence: a modern approach, Prentice Hall, 2020.
- [6] R. Parikh, Knowledge and the problem of logical omniscience., in: ISMIS, volume 87, Citeseer, 1987, pp. 432–439.

can beaten by deriving general rules with as many occurrences of right in them as possible and combining such a rule with book(a) as late as possible.

- [7] M. Wang, Y. Tang, J. Wang, J. Deng, Premise selection for theorem proving by deep graph embedding, arXiv preprint arXiv:1709.09994 (2017).
- [8] D. Huang, P. Dhariwal, D. Song, I. Sutskever, Gamepad: A learning environment for theorem proving, in: International Conference on Learning Representations, 2018.
- [9] S. Polu, I. Sutskever, Generative language modeling for automated theorem proving, arXiv preprint arXiv:2009.03393 (2020).
- [10] M. Rawson, G. Reger, A neurally-guided, parallel theorem prover, in: International Symposium on Frontiers of Combining Systems, Springer, 2019, pp. 40–56.
- [11] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (2015) 529–533.
- [13] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.