Folding Architecture Networks Improve the Performance of Otter

Michael Blanchard[1], Joseph D. Horton[1], Dawn MacIsaac[1]

[1] University of New Brunswick, Faculty of Computer Science,
Fredericton, Canada
s11r@unb.ca, jdh@unb.ca, dmac@unb.ca

Abstract. A neural network (folding architecture network) supplements the standard symbol count heuristic of Argonne National Laboratory's theorem prover, Otter. The network is trained to differentiate between clauses that are likely to be used in a proof and those not likely to be used in a proof. The result is the network enabled version of Otter finds more proofs among the TPTP (Thousands of Problems for Theorem Provers) problem set than the original version of Otter. The optimal configuration in this work yielded a net gain of 29 proofs which according to McNemar's statistical test of proportions, is strongly significant ($\alpha = 0.05$, $p < 0.002$).

Keywords: Folding Architecture Network, Otter, Automated Reasoning, Resolution Theorem Prover, Neural Networks, given-clause algorithm, TPTP

1 Introduction

The given-clause algorithm, used by resolution style theorem provers, has a high branching factor. Better clause selection heuristics reduce the number of given-clauses selected to find a proof, and over a large set of problems, the number of proofs found should increase with improved heuristics.

We use a folding architecture (FA) network [2, 8, 6] to learn from past proof experiences encoded as annotated clause patterns [7] to improve the clause selection heuristic of Otter [4]. FA networks are a type of artificial neural network capable of processing recursive data structures such as first order logic clauses. Annotated clause patterns are abstract representations of clauses annotated with statistics that describe how useful the underlying concrete clauses have been in finding proofs.

2 Background

This work can be viewed as an extension of work completed by Goller [2] and Schulz [7]. Goller used FA networks to learn heuristic evaluation functions for the automated theorem prover, SETHEO [3]. He showed that a FA network can be used

to improve the performance of an automated theorem prover. However, since testing was restricted to a single domain, the results are preliminary.

Specifically, Goller trained several FA networks on training data collected from 300 solved problems from the domain encapsulating word problems in group theory. Testing was performed on a separate set of 19 problems.

The best of the heuristics acquired by the FA networks allowed SETHEO to solve 17 of the 19 testing problems with an average search time of 2.9 seconds. The original version of SETHEO was not able to solve any of the 19 test problems within the 80 seconds allotted, but a version of SETHEO (E-SETHEO) with special equality handling was able to solve the same 17 problems with an average search time of 5.5 seconds.

Schulz [7] developed a framework for learning search control heuristics based on past proof experiences. A database populated with past search decisions and their results is the major component of the framework. All given-clauses are converted into abstract clause patterns and associated with statistics that describe their usefulness in finding the proof.

He used the machine learning technique, ten fold stratified cross validation, which partitions the problem set into 10 equal-sized folds. Each fold is tested using the other nine folds as training data.

Schulz' approach improves the performance of the given-clause algorithm compared to the standard symbol count heuristic alone. The superposition-based prover, E, found 7% more proofs when modified by past proof experiences. His results confirm that some patterns do correspond to contributing clauses, while others correspond to superfluous clauses, and that incorporating annotated patterns into the given-clause algorithm improves overall performance.

3 Experimental Setup

The performance of Otter was improved by using FA networks trained on annotated clause patterns. The experiments used ten fold stratified cross validation. The regular version of Otter was run for 60 seconds on each of 6056 problems from Thousands of Problems for Theorem Provers Problem Library (TPTP)[1] [9]. From the set of 2253 problems that were solved, all unit given clauses with no more than ten symbols were converted into abstract annotated clause patterns as per Schulz.

These patterns were then classified as one of three types: contributing, superfluous, and ambiguous. Contributing patterns are patterns that frequently correspond to clauses used in proofs. Superfluous patterns are those that frequently correspond to clauses not used in proofs. Finally, ambiguous patterns are those that cannot confidently be classified as either contributing or superfluous.

The classification of a pattern was accomplished using an assumption that a pattern and its associated statistics are the aggregation of a series of Bernoulli trials. The motivation for this treatment is the observation that the probability of any particular clause being used in the proof is a discrete binomial random variable. A

[1] The results reported here were obtained in compliance with the guidelines of the TPTP. The TPTP release number is 3.1.0

clause is either used in the proof or it is not (e.g. it is contributing or superfluous). Therefore, for every pattern, p, there can be calculated a proportion, $ContributionRate_p$, that is the proportion of all observations the pattern corresponded to a contributing clause. The calculation of a confidence interval for a proportion is straight forward, and is not presented here [5]. Similarly, a population-wide statistic for the proportion of observations corresponding to a contributing clause (denoted $ContributionRate_{pop}$) and an associated confidence interval can be calculated for the entire set of all unit clauses of no more than 10 symbols. $ContributionRate_{pop}$ corresponds to the probability that a unit clause of no more than 10 symbols chosen at random from the set of given clauses is contributing. $ContributionRate_p$, $ContributionRate_{pop}$, and their respective confidence intervals together form the basis by which a pattern is classified.

The FA network is trained as a classifier using the contributing and superfluous patterns as training data. The trained FA network supplements the standard symbol count heuristic of Otter. Two types of experiments were performed: two-way classification and three-way classification.

Two-way classification adds a small fixed amount to the standard heuristic for those clauses that are classified as either ambiguous or superfluous. Three-way classification adds a small fixed amount to the standard heuristic for those clauses that are classified as ambiguous, and adds a little more to those clauses that are classified as superfluous. For a detailed discussion, the reader is referred to [1].

4 Results

Fourteen experiments were conducted using almost all the domains of TPTP. Five experiments used two-way classification, and nine used three-way classification. Other parameters varied were how the classifications were done, the size of the pick-weight adjustment [4], and the number of neurons in the FA network. One experiment (Experiment #11) substituted a hash table for the FA network. Table 1 lists the results of the 13 FA network experiments.

Table 1 FA Network Experimental Results Relative to Regular Otter

Experiment #	Network ID	New Proofs Found	Old Proofs Missed	Net	McNemar's Test P Value
1	*FA-1*	53	23	+30	0.000765
2	*FA-1*	70	43	+28	0.0114
3	*FA-2*	50	26	+24	0.00791
4	*FA-3*	46	29	+17	0.0639
5	*FA-4*	48	26	+22	0.0141
6	*FA-4*	51	23	+28	0.00152
7	*FA-4*	47	27	+20	0.0265
8	*FA-4*	52	29	+23	0.0140
9	*FA-1*	52	27	+25	0.00655
10	*FA-5*	50	30	+20	0.0330
12	*FA-6*	51	27	+24	0.00877
13	*FA-7*	55	26	+29	0.00169
14	*FA-8*	48	30	+18	0.0535

All but two experiments (#4 and #14) showed a statistically significant improvement according to McNemar's test ($\alpha = 0.05$, $p < 0.034$), which tests whether or not two proportions of paired binary data are different. To test the performance of a memory based system, an experiment was conducted using a hash table substituted for the neural network. Table 2 contains the results of comparing the hash table version of Otter to the regular version of Otter and to the FA version of Otter. Note the statistically significant increase in performance in both cases according to McNemar's test ($\alpha = 0.05$, $p < 0.009$).

Table 2 Hash Table Implementation Compared to Regular Otter & Network Version #13

Compared To	New Proofs Found	Old Proofs Missed	Net	McNemar's Test P Value
Regular Otter	75	24	+51	2.77×10^{-7}
FA Net Otter #13	43	21	+22	0.00815

5 Conclusions

The purpose of this work was to determine whether or not a FA network can be used to improve the performance of a resolution style automated theorem prover. We successfully showed that the FA version of Otter outperforms the regular version of Otter. In addition, we showed that the hash table version of Otter not only outperforms the regular version of Otter, but it also outperforms the best of the FA versions of Otter.

The results of the experiments where indicate that the FA network enabled Otter outperforms the regular version of Otter. The result of the hash table experiment leaves no doubt that the hash table implementation of Otter is better than regular Otter. Not only is it better than regular Otter, it is better than the network enabled Otter too (McNemar's test $\alpha = 0.05$, $p < 0.009$).

The fact that the hash table implementation performs best probably means that memorization of known patterns is what is responsible for the performance improvement, rather than the ability to generalize the knowledge to new patterns. However, one should keep in mind that the performance differential between the FA network and the hash table could also mean that the cost of erroneously classifying a pattern is high. The hash table does not make mistakes, but the FA network does. It is possible that such errors are especially costly. Perhaps the performance differential is due to both these effects.

In any case, the artificial neural network technology itself seems to add very little if any to the performance improvement and it is in fact the abstract clause patterns that are doing the work. It is possible that abstract clause patterns are too abstract for a neural network to notice anything that would give it an edge over a pure memory based system such as the simple hash table. Perhaps the neural network needs more specific features; either about the clause itself, or specific features about the context in which the clause exists.

References

1. Blanchard, M.: Folding Architecture Networks Improve the Performance of a Resolution Style Automated Theorem Prover, Master's Thesis, Faculty of Computer Science, University of New Brunswick, 2006.
2. Goller, C.: Learning Search-Control Heuristics for Automated Deduction Systems with Folding Architecture Networks. ESANN'1999 proceedings – European Symposium on Artificial Neural Networks Bruges (Belgium), pages 45-50, 21-23 April 1999.
3. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: SETHEO: A High-Performance Theorem Prover. Journal of Automated Reasoning, 8(2):183-212,1992.
4. McCune, W. Otter 3.3 Reference Manual. Argonne National Laboratory, Mathematics and Computer Science Division, 2003.
5. Milton, J. S., Arnold, J. C.: Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences, 3rd Edition. Irwin/McGraw-Hill, 1995.
6. Schmitt, T. Evaluation of the Neural Folding Architecture for Inductive Learning Tasks concerning Logical Terms and Chemical Structures. Master's Thesis, Technical University of Munich, Computer Science, 1997.
7. Schulz, S. Learning Search Control Knowledge for Equational Deduction. Number 230 in DISKI. Akademische Verlagsgesellschaft Aka GmbH Berlin, 2000. Ph.D. Thesis, Fakultat fur Informatik, Technische Universitat Munchen.
8. Schulz, S., Kuchler, A., Goller, C.: Some Experiments on the Applicability of folding Architecture networks to guide Theorem Proving. In D.D> D ankle II, editor, Proceedings of the 10th FLAIRS, Daytona Beach, pages 377-381. Florida AI Research Society, 1997.
9. Sutcliffe, G. and Suttner, C.B., The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning, Vol 21, No 2, pp. 177-203, 1998.