In *Proceedings of the The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN2000)*
May 11-13, 2000.

Neuro-Evolution and Natural Deduction

Nirav S. Desai
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712-1188
ndesai@cs.utexas.edu

Risto Miikkulainen
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712-1188
risto@cs.utexas.edu

Abstract- Natural deduction is essentially a sequential decision task, similar to many game-playing tasks. Such a task is well suited to benefit from the techniques of neuro-evolution. Symbiotic, Adaptive Neuro-Evolution (SANE)(Moriarty and Miikkulainen 1996) has proven successful at evolving networks for such tasks. This paper will show that SANE can be used to evolve a natural deduction system on a neural network. Particularly, it will show that (1) incremental evolution through progressively more challenging problems results in more effective networks than does direct evolution, and (2) an effective network can be evolved faster if the network is allowed to "brainstorm" or suggest any move regardless of its applicability, even though the highest-ranked valid move is always applied. This way evolution results in neural networks with human-like reasoning behavior.

1 Introduction

Many of the successes in theorem proving have been achieved through symbolic systems (Boyer and Moore 1975; Anderson 1983; Newell 1980b). Most of these symbolic theorem provers depend on some decision procedure particular to the logic on which they operate. They must restrict the logic to a decidable subset of first-order logic, such as propositional logic or Horn logic. Others, like the well-known resolution method (Robinson 1963), perform in an unintuitive manner. While these methods are complete for decidable logics, they do not scale up to more complicated logics.

Natural deduction systems are heuristics-based methods that prove theorems using inference rules such as modus ponens, disjunctive syllogism, and modus tollens. They create proofs in an intuitively human-like manner. They rely on heuristics in choosing which rules to apply. While decision procedures are particular to the logic in question, heuristics are more general and scalable to more complex logics.

Neural networks are effective for pattern recognition tasks and can be used to implement heuristic proof decisions. Genetic algorithms learn from sparse feedback and can perform credit assignment even when the correct behavior is unknown. This paper shows how these two methods can be combined to form a natural deduction system that learns and reasons similarly to humans.

2 The Domain of Theorem Proving

Back-chaining is a goal-driven method that provides a natural platform for studying human-like theorem proving. In back-chaining, the prover starts with the theorem and determines what information is needed to prove it. It proceeds recursively until it arrives at an axiom. For example, given the axioms:

1. a
2. $a \rightarrow b$
3. $b \rightarrow c$

and "b" as the theorem, we first use modus ponens on axiom 2 to find out that we need to prove a. Since axiom 1 is a we have found the full proof: $(a \wedge (a \rightarrow b)) \rightarrow b$.

Back-chaining is easily characterized as a sequential decision task. Such tasks require multiple steps to arrive at a solution and provide only sparse reinforcement. Credit must be assigned for each step even though there is no way of knowing immediately if the step is correct. The system will only know if it followed the correct path at the end of the process. For this reason, there needs to be some kind of heuristic mechanism to determine the benefits of each decision, i.e. to perform credit assignment. Neuro-evolution is such a method.

3 Symbiotic, Adaptive Neuro-Evolution

Most neuro-evolution methods evaluate and evolve whole networks. If a network performs well in a given environment, it is rated highly and recombined with other good networks to produce offspring networks. If it does not, it receives a low fitness value and may be removed from the population.

However, if neuro-evolution is done at the level of partial solutions, the process turns out much more effective. This idea is used in SANE (Moriarty and Miikkulainen 1996). SANE evolves two separate populations, one of nodes and another of network blueprints. The node population focuses on developing specific problem-solving functionality for the given task. The network population focuses on combining the nodes effectively.

The key to successful evolution is maintaining a diverse population. In SANE, nodes are assigned fitness values based on the networks in which they participate. If the same node is used in many good networks it will be rated highly. Just as

In *Proceedings of the The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN2000)*
May 11-13, 2000.

a team will work well if it has players that can work well together, a network will be effective if it has nodes that can cooperate. By evolving nodes that come together to form networks, SANE maximizes diversity and enables an efficient search of the solution space (Moriarty and Miikkulainen 1996).

SANE has proven to be more effective than other neuro-evolution and reinforcement learning techniques in several domains, including the pole-balancing benchmark (Moriarty and Miikkulainen 1996). It also provides a general learning algorithm that can be reused in many domains, which makes it a good candidate for implementing natural deduction.

Two discoveries made in prior work with SANE are particularly relevant for the natural deduction domain: incremental evolution and brainstorming. In the usual direct evolution, each network in the population is evaluated based on the full desired functionality in the task. In incremental evolution (Gomez and Miikkulainen 1997), the population is evolved to achieve a subset of the full desired functionality, and gradually more functionality is added. For example, in the direct approach, networks would be evolved both to find relevant rules and to come up with short proofs. In the incremental approach, networks would first be evolved to make an inference for a given step in a proof. Then, in the second stage, the fitness function would be modified to include minimizing the number of steps. This way, the environment is gradually made more challenging, culminating in the complete task.

In brainstorming the network suggests inferences regardless of whether they can be applied (Moriarty and Miikkulainen 1995). However, only the valid inference with the highest activation is ever executed. Because diversity is the key to successful evolution, any restrictions on behavior can reduce diversity and risk harming the performance of the algorithm. Brainstorming, by reducing restrictions, allows exploring a wider set of behaviors and results in a more effective search.

Incremental evolution and brainstorming are very human-like methods of learning, and will be shown to be beneficial also in artificial evolution for theorem proving.

4 Hypotheses

This paper will show that a natural deduction system can be evolved on a neural network. The goal is not to implement all rules of natural deduction, only to show that neuro-evolution is capable of evolving networks with heuristics for the most central set of rules, i.e. modus ponens and modus tollens, and how to realize when the proof is complete. If the network can prove transitivity, it will show understanding of modus ponens. Thus, given $a, a \rightarrow b, b \rightarrow c$, it must be able to show c. If it can prove negations, it understands modus tollens: given $(a \rightarrow b) \wedge \neg b$ it must be able to show $\neg a$. A system that shows such a facility with modus ponens and modus tollens will theoretically be extensible to a full natural deduction system.

In order to verify network performance, this experiment will focus on propositional logic. However, a network that learns this method will be equally usable to solve first-order logic theorems provided it can be extended with rules for reasoning with quantification symbols.

This experiment was designed to test the following three hypotheses:

1. A neural network can be evolved to perform natural deduction.
2. Incremental evolution will produce more effective networks than would be produced by evolving all functionality at the same time.
3. Allowing brainstorming results in more effective evolution than requiring the network to always suggest only valid inferences.

5 Procedure

In this section, network configuration and encoding will be described. The experiment will be discussed in three subsections. Section 5.2 will report on the evolution of networks in three separate populations to infer when to apply either QED, MP, or MT. Section 5.3 combines these three populations to evolve networks that can handle all three rules. Section 5.4 takes the final population from section 5.3 and evolves it further to create multi-step proofs. This way, full functionality is evolved incrementally.

5.1 Representation and Configuration

The first step is to configure the parameters for the networks that will be generated by SANE, which includes deciding on a number of input and output nodes, and what these nodes will represent.

We used a three-layer neural network architecture with 24 input, 120 hidden, and 9 output nodes as shown in figure 1.

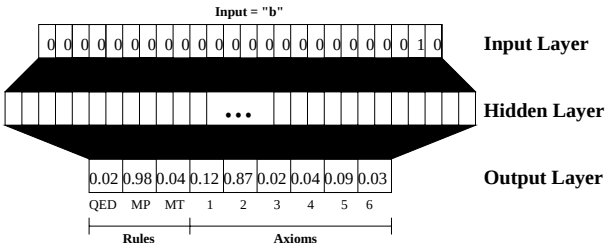

Figure 1: Network architecture and behavior given the input of "b".

Networks are evolved to prove theorems for a particular set of axioms. In the current experiment we used the following set of axioms:

1. a

In *Proceedings of the The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN2000)*
May 11-13, 2000.

2. $a \rightarrow b$
3. $b \rightarrow c$
4. $a \rightarrow c$
5. $d \rightarrow \neg c$
6. $d \rightarrow \neg a$.

The network's input represents the theorem to be proven, and the output indicates the axiom to be used and the rule that is applied to that axiom. The rule is applied and the theorem that needs to be shown next is determined outside the system. Thus, if "b" is given as the input theorem, the network proves it in two steps, generating the following output:

1. modus ponens on axiom 2
2. QED, axiom 1.

The first step of this proof is demonstrated in figure 1.

A localist 24-node encoding scheme was used to input the representations of theorems. There are three groups of eight nodes in this encoding. Each group of eight nodes represents a literal or a binary operator. The group on the left and the right can represent either a, b, c, d, $\neg a$, $\neg b$, $\neg c$, or $\neg d$. The middle eight can represent $\rightarrow$, $\wedge$, or $\vee$ ($\wedge$ and $\vee$ are not used in this experiment but are included for future extensions). The rightmost unit in the left and right groups stands for the character a, the one next to it b, then c and d. The fourth unit from the left represents negation. Thus, $[00010010] = \neg b$. The three leftmost nodes are currently not used. In the middle group, the leftmost unit indicates an implication. Figure 2 provides some examples of how theorems are encoded.

$$\begin{aligned} b = &\quad [00000000\ 00000000\ 00000010] \\ a \rightarrow b = &\quad [00000001\ 10000000\ 00000010] \\ \neg c \rightarrow \neg d = &\quad [00010100\ 10000000\ 00011000] \end{aligned}$$

Figure 2: Examples of theorem encoding.

Output is encoded with nine nodes. The first three nodes indicate the rules QED, MP, and MT. The remaining six each represents a different axiom. The network will activate the output node for the rule it chooses to apply and the axiom to which this rule is applied. Figure 1 shows the output of a network suggesting the application of MP on axiom 2. The MP node and the axiom 2 node have high activation while all other nodes have low activation.

With the current encoding scheme, 72 different inputs can be represented. The four atoms and their negations constitute eight of them. The remaining 64 consist of all possible implications between these eight primitives. This encoding scheme allows representations to be distinct. However, it uses many input (24) and output (9) nodes and requires many hidden nodes (120) to achieve satisfactory performance, which slows learning and performance. In this domain, though, the advantages outweigh the disadvantages.

During evolution, the fitness values of the different networks were evaluated based on how they performed when the six different axioms, "$\neg b$", "b", "$\neg d$", and nine other randomly selected theorems (out of the total 72) were given as input. The first six were selected to evolve QED functionality. The next three were selected to evolve MP and MT functionality. The remaining nine were selected randomly at each evolution to achieve good generalization.

5.2 Part I: Three Distinct Populations

The first goal was to evolve three populations: (1) one population to detect the end of a proof (QED), (2) another population to detect when to apply modus ponens (MP), (3) the last to detect when to apply modus tollens (MT).

A QED network checks to see if the input theorem is an axiom. If so, the network activates the output node for the axiom that matches the theorem and the rule node for QED. For example, if the input is "a" and axiom 2 is also "a", the QED output node and the axiom 2 output node should have high values. All other output nodes should have low values.

An MP network checks to see if there is a premise that has the input theorem as the consequent. For example, if the input theorem is "c" and axiom 3 is "$b \rightarrow c$", the MP output node and the axiom 3 output nodes should have high values. The remaining theorem to be shown is "b".In other words, MP can be used with premises of the form $x \rightarrow y$ where y is the theorem.

An MT network checks to see if there is a premise of the form $\neg x \rightarrow y$ where x is the given input theorem. For example, if the input theorem is "c" and axiom 3 is "$\neg c \rightarrow a$", the MT output node and the axiom 3 output nodes should have high values. The remaining theorem to be shown then is "a".

In order to apply SANE to this task, the experimenter needs to write an evaluation function that assigns fitness values to networks. The evaluation function must rate those networks highly that give the desired output, and rate those networks poorly that do not.

Let node c be the node corresponding to the correct axiom, d be the node corresponding to the rule to apply, n be the number of output nodes, and out_i be the activation of node i. The evaluation function returns the following value.

$$\text{Fitness} = out_c + out_d - \sum_{i=0; i \neq c, i \neq d}^{n} out_i. \qquad (1)$$

In evolving the QED population, $d = 0$ since node 0 of the output layer is the node representing QED (figure 1), c is the node representing an axiom that is equivalent to the given theorem. For MP, $d = 1$ and c is the node representing an axiom of the form $r \rightarrow s$ where s is the given theorem. For MT, $d = 2$ and c is the node representing an axiom of the form $\neg r \rightarrow s$ where r is the given theorem.

In *Proceedings of the The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN2000)*
May 11-13, 2000.

5.3 Part II: The Combined Single-Step Network

The next goal was to combine the above three network populations into a single population and evolve a network to perform well on all three rules. This evolution was done both incrementally, using populations from section 5.2 and directly by starting over with a new random population.

When the QED, MP, and MT fitness functions are combined, the evolution should generate a network that is capable of using all three rules. Therefore, each network evaluation consisted of three components: all variations of QED, four randomly chosen examples of MP, four randomly chosen examples of MT. The component fitness values were summed and divided by the number of examples to get the final network evaluation.

For both the direct and incremental evolution experiments, two distinct network populations were evolved to investigate brainstorming. One was evolved with a fitness function that allowed brainstorming, the other with a fitness function that penalized for invalid inferences. Thus, in the brainstorming case, if the network suggested both MP and MT when only MP applied, it was not penalized. Thus, the network could suggest invalid moves, but only the highest valid move was actually applied.

5.4 Part III: The Multi-step Network

The third goal was to evolve a network able to generate short multi-step proofs using QED, MP, and MT. To do this, we need an evaluation function that assigns fitness values only after a series of inferences. At this point the problem becomes a true sequential-decision task. Again, four different evolution experiments were run: directly from a random population and incrementally from the final population of section 5.3, both with and without brainstorming.

If the network failed to complete the proof within a certain number of steps, its fitness was proportional to the number of rules and axioms for which it inferred correctly. If a network got all the way through the proof, choosing correct rules and axioms at each step, it was given a fitness inversely proportional to the number of rules and axioms it inferred correctly, then doubled to ensure that successful proofs receive a higher fitness than failed proofs.

6 Results

This section will present the results of the experiments described above. Each experiment was performed ten times with different random seeds so that statistical error could be calculated.

6.1 Part I: Three Distinct Populations

In Part I, separate networks were evolved for QED, MP, and MT. The QED evolution performed very well. The final network was tested on each of the six axioms, and the network selected the rule QED correctly in every case.

The final network evolved in the MP population was tested on the theorems "a", "b", "c", "$\neg d$" and four other randomly selected theorems of the form $x \rightarrow y$, namely "$b \rightarrow d$", "$a \rightarrow \neg b$","$d \rightarrow c$", and "$\neg a \rightarrow \neg d$". The MP network also performed exceedingly well. In all test cases it produced the proper output. For example, when the theorem "b" was the given as input, the network activated output nodes for premises of the form $x \rightarrow b$. Also, when given "c" the net inferred to use MP on axioms 3 and 4, even though "c" was never presented during evolution as a theorem.

The final network of the MT population was also tested on the theorems used to test MP. The MT evolution also performed well. The final network was able infer correctly for each theorem tested. For example, given the theorem "$\neg d$", the final network was able to recognize that MT could be used on axiom 5 and 6.

6.2 Part II: The Combined Single-Step Network

The combined networks were tested on all the test inputs from Part I. In direct evolution, where the population was started with random values and all rules were evolved at once, evolution had trouble finding networks that could use all three rules. The networks performed correctly for some inputs and not for others. Even prolonged evolution did not help. However, in the incremental evolution, where the three final populations from section 5.3 were combined and evolved further, the final network worked for all test cases.

Because the direct evolution failed, brainstorming could be tested only the incrementally evolved population. On the usual set of axioms, performance of brainstorming evolution was not statistically significantly different from the non-brainstorming evolution. Both were able to infer correctly.

However, on a separate evolution with different axioms we had some interesting results. A population was evolved with the axioms "a", "$a \rightarrow b$", "$b \rightarrow a$" and "$b \rightarrow c$". It was then asked to prove the theorem "a". This is an example of an ambiguous step because either QED on axiom 1 or MP on axiom 3 can be applied. The brainstorming network was unable to give a confident suggestion on what to do in this situation. It was able to suggest QED but could not decide which axiom to apply. On the other hand, in a non-brainstorming evolution with the same axioms, the network was not confused because the evaluation function explicitly ordered QED above MP and MT. The network was explicitly penalized if it tried to apply MP or MT before QED. Therefore the final network knew to apply QED to axiom 1, not axiom 3, because QED only applies to axiom 1.

The final networks of both the brainstorming and non-brainstorming evolutions were able perform well on all test cases with the usual set of six axioms. The problem in the second set is interesting because it identifies a potential pitfall of brainstorming: looser constraints may lead to networks that cannot decide between alternatives. Interestingly, this problem does not arise when the networks are evolved to perform multi-step proofs, as will be discussed next.

In *Proceedings of the The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN2000)*
May 11-13, 2000.

6.3 Part III: A Multi-step Network

As mentioned in section 5.1, networks in Part III were evaluated based on the same set as in Part I and Part II, but now network fitness for evolution was calculated only after the proof was completed. Direct evolution was again not very effective because it was very difficult to get started on proofs. For example, when "b" is the input theorem, the solution is a two-step proof. The second step will only be reached if the network makes the correct inference on step one. However, it does not get to the end of the proof, and always gets a poor fitness. Thus the networks do not even learn to perform the first step of multi-step proofs.

In incremental evolution, the best network population from section 6.2 for single step proofs was used as a starting point, but now evaluated on their performance on multi-step proofs. Thus, the basic theorem proving functionality was evolved first and then extended into a more challenging task. This method resulted in successful networks. Not only was the best network able to complete the proof in most situations, it was able to minimize the number of steps. If given "c", the network was able to infer MP on axiom 4 rather than MP on axiom 3, which would result in a longer proof. Recall that the population never saw "c" as input during evolution; the network was able to generalize and optimize. The incremental approach succeeded where the direct method failed.

We then tested the performance of the final networks evolved with and without brainstorming. Both networks performed well in all test cases. They were both able to produce the shortest possible proofs for the theorems tested. Thus, there was no significant qualitative difference in the performance of the two networks. However, the brainstorming evolution was on the average $6.35 \pm 1.05\%$ faster than non-brainstorming evolution, based on ten trials with different random seeds. Most importantly, it was able to use the goal of generating short multi-step proofs to solve the ambiguity problem in section 6.2. The loosening of restrictions during evolution did not diminish overall performance, but rather, led to an overall speedup in training. Such a result is a good indication of the power of brainstorming evolution.

7 Further Research

This experiment is a good starting point for further research into evolving human-like natural deduction systems. It shows that such a theorem prover can be evolved in a limited domain. However, it also prescribes a procedure for how to scale to larger domains. In order to create a general purpose theorem prover we need to include more rules. While logically all propositional logic can be proven using negation and modus ponens, we are interested in intuitive human-like natural deductions systems, and need rules that humans would use, such as those dealing with with conjunctions and disjunctions. For conditional proofs it is necessary to be able to include assumptions in the axiom list, which means the axioms must be explicitly represented. The network also needs some type of memory to distinguish actual axioms from the assumptions. We also need common sense rules to identify tautologies like "$a \rightarrow a$." After these extensions, the theorem prover could be extended to first-order logic.

Another important result that should be investigated further is that neuro-evolution can be used to model human learning and reasoning. Rules were learned one at a time and incorporated into a growing body of knowledge. Internal representations of axioms were developed through evolution of localist output only. Finally, heuristic strategies were developed for using rules of inference. With the neuro-evolution approach, we can learn much about the process of reasoning on rational systems in general, including that of humans.

8 Conclusions

This experiment showed that neuro-evolution can be used to learn natural deduction. More specifically, SANE was successful at evolving networks that, through back-chaining, create multi-step proofs using modus ponens and modus tollens. Incremental evolution through a series of gradually more difficult tasks was more effective at producing successful networks than direct evolution. The experiment also showed that brainstorming allows a genetic algorithm to evolve networks faster by enforcing fewer constraints. These results constitute a promising starting point for developing comprehensive natural deduction systems through neuro-evolution.

Bibliography

Anderson, J.R. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.

Boyer, R.S. and Moore, J.S. (1975). Proving theorems about LISP functions. *Journal of the Association for Computing Machinery*, 22:129–144.

Chang, C.L. and Lee, R. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, NY.

Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

Moriarty, D.E. and Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7:195–209.

Moriarty, D.E. and Miikkulainen, R. (1996). Efficient Reinforcement Learning through Symbiotic Evolution. *Machine Learning*, 22:11-32.

Moriarty, D.E. and Miikkulainen, R. (1998). Forming Neural Networks through Efficient and Adaptive Coevolution. *Evolutionary Computation* 5:373-399.

Newell, A. (1980a). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In N. Nickerson, editor, *Attention and Performance VIII*, pages 693–718. Lawrence Erlbaum Associates, Hillsdale, NJ.

Newell, A. (1980b). Physical symbol systems. *Cognitive Science*, 4:135–183.

Robinson, J.A. (1963). Theorem-Proving on the Computer. *Journal of the ACM*, 10:163–174.