Neural Logic Network Learning using Genetic Programming

Chew Lim Tan, Henry Wai Kit Chia
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{tancl, chiawaik}@comp.nus.edu.sg

Abstract

Neural Logic Network or *Neulonet* is a hybrid of neural network expert systems. Its strength lies in its ability to learn and to represent human logic in decision making using component *net rules*. The technique originally employed in *neulonet* learning is backpropagation. However, the resulting weight adjustments will lead to a loss in the logic of the *net rules*. A new technique is now developed that allows the *neulonet* to learn by composing *net rules* using genetic programming. This paper presents experimental results to demonstrate this new and exciting capability in capturing human decision logic from examples. Comparisons will also be made between the use of *net rules*, and the use of standard boolean logic of negation, disjunction and conjunction in evolutionary computation.

1 Introduction

There has been a fair amount of interest in training neural networks using genetic programming [Golubski and Feuring, 1999]. Gaudet [1996] applied genetic programming on logic-based neural networks which basically represent standard logic (negation, conjunction and disjunction). Similarly, an adaptive logic network [Armstrong and Thomas, 1996] uses linear threshold units at the leaf node level, while the parent nodes of the network are composed of "AND" and "OR" logic units. However, neural networks can also represent non-standard logic. A Neural Logic Network or simply *Neulonet* has been proposed that emulates human decision logic which are often too complex to be expressed neatly using standard logic [Teh, 1995; Tan *et al.*, 1996].

In this paper, we begin by focusing on the integration of *neulonets* into a genetically programmed environment. First, we introduce the concept of *neulonets* and show how they can be combined to form complex decisions. We then describe a GP system for evolving *neulonets*, and discuss how genetic operations of crossover and mutation can be adapted to *neulonet* evolution. Next, we demonstrate how the system can effectively solve a classification problem and extract rules from the evolved *neulonet*. Finally we provide experimental results to substantiate the use of *neulonets* as an improvement over the use of standard logic networks in genetic programming.

2 Neural Logic Network

A *Neulonet* has an ordered pair of numbers associated with each node and connection (figure 1). Let Q be the output node and $P_1, P_2, \ldots P_N$ be input nodes. Let values associated with the node P_i be denoted by $(a_i,\ b_i)$, and the weight for the connection from P_i to Q be $(\alpha_i,\ \beta_i)$. The ordered pair for each node takes one of three values, namely, (1,0) for true, (0,1) for false and (0,0) for "don't know". (1,1) is undefined. The following activation function determines the output at Q:

$$Act(Q) = \begin{cases} (1,0) & \text{if } \sum_{i=1}^{N}(a_i\alpha_i - b_i\beta_i) \geq \lambda \\ (0,1) & \text{if } \sum_{i=1}^{N}(a_i\alpha_i - b_i\beta_i) \leq -\lambda \\ (0,0) & \text{otherwise.} \end{cases} \quad (1)$$

where λ is the threshold, usually set to 1.

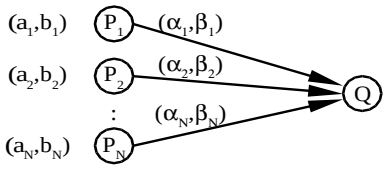

Figure 1: A basic Neural Logic Network - *Neulonet*.

By applying equation (1) on a network in figure 2, the network behaves like an expert system rule with an OR logic operation as shown in the truth table.

P	Q	P or Q
(1,0)	(1,0)	(1,0)
(1,0)	(0,1)	(1,0)
(1,0)	(0,0)	(1,0)
(0,1)	(1,0)	(1,0)
(0,1)	(0,1)	(0,1)
(0,1)	(0,0)	(0,0)
(0,0)	(1,0)	(1,0)
(0,0)	(0,1)	(0,0)
(0,0)	(0,0)	(0,0)

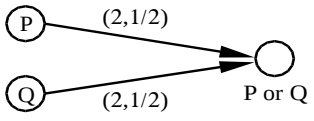

Figure 2: The above *neulonet* behaves like a disjunction rule.

We call a network in figure 2 a *net rule* – a rudimentary network that can be chained with other similar networks just like in a conventional rule-based expert system. The power of a *net rule*, however, is not limited to standard boolean logic operations. A wide range of different human logic in decision making can be represented with *net rules* using different sets of weights. Figure 3 shows some examples of *net rules* that emulate a human decision maker's behavior. Human decision processes are often too complex to be expressed neatly using standard logic. One such complexity is that human reasoning can be biased by giving different degrees of importance to different factors. A simple AND or OR operation will be inadequate in representing varying degrees of bias.

For instance, rule (4) "Priority" in figure 3 exhibits different degrees of influence on the outcome of Q with P_1 being the most important factor, while P_n will only be considered when all other factors are unknown, i.e. (0,0). Another example can be seen in rule (13) "Silence means consent" where *Def* represents the default action. If no one among P_1, P_2, ..., P_n has any opinion, or if there is an equal number of persons either for or against a motion, the default action is taken. In all other cases, the majority wins. *Net rules* can combine to form composite *net rules* to achieve more complicated decisions. For instance, using the output of rule (7) "Unanimity" as the input V to rule (10) "Veto", both with *n* set to 2, an XOR operation can be realized as shown in figure 4.

A *neulonet* combines the strengths of neural networks and expert systems [Tan *et al.*, 1996]. It was proposed that a knowledge engineer could first encode his knowledge into component *net rules* and later use real examples to do refinement training to adjust the weights in the *neulonet*. Training in this case is by means of backpropagation [Teh, 1995], which will pervasively modify all weights in the network. It was shown that the *neulonet*'s performance improves after the refinement training. However, one problem with backpropagation is that after training, the logic of the *net rules* may not be decipherable as the weights of the *net rules* have been altered. The trained *net rules* may not be reused easily. In view of this, a novel way of *neulonet* training by means of a genetic programming paradigm is now introduced.

3 Genetic Programming

Genetic programming [Koza, 1992] is an extension of the conventional genetic algorithm where instead of subjecting bit patterns to genetic evolution, the individuals in the gene population are computer programs. In the context of *neulonet* evolution, the problem statement may be stated as follows:

> "Given a set of input nodes, a library of *net rules*, an output node, and a set of examples containing instances of input values together with the corresponding output decision, apply genetic programming to construct a *neulonet* that represents a decision logic induced from the examples".

The solution may be carried out in three steps:

1. Generate an initial population of *neulonets* comprising of random compositions of *net rules* according to the available input nodes and the output node.

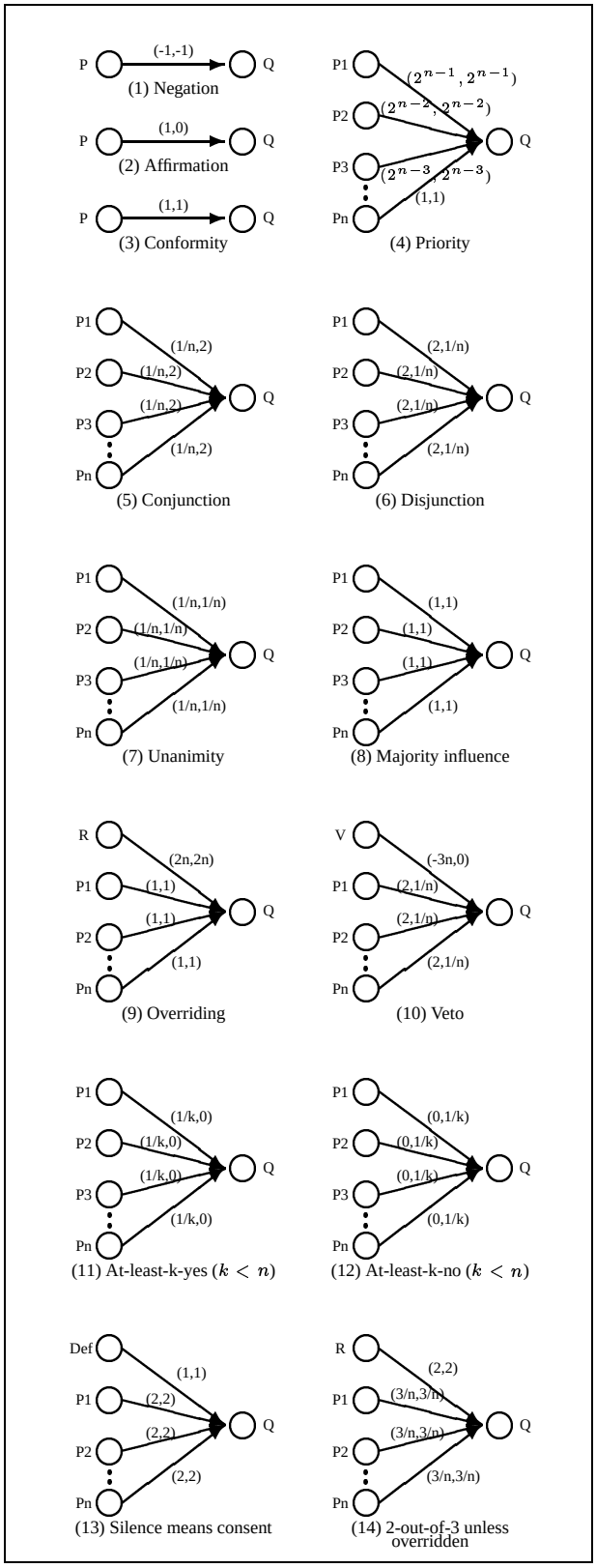

Figure 3: Examples of component *net rules*.

P	Q	P xor Q
(1,0)	(1,0)	(0,1)
(1,0)	(0,1)	(1,0)
(1,0)	(0,0)	(1,0)
(0,1)	(1,0)	(1,0)
(0,1)	(0,1)	(0,1)
(0,1)	(0,0)	(0,0)
(0,0)	(1,0)	(1,0)
(0,0)	(0,1)	(0,0)
(0,0)	(0,0)	(0,0)

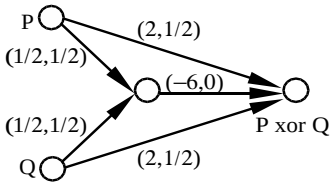

Figure 4: An XOR composite *net rule*.

2. Iteratively perform the following substeps until the termination criterion has been satisfied:
   (a) Fire each *neulonet* in the population and assign it a fitness value using the fitness measure.
   (b) Create a new population of *neulonets* by applying the following three operations to *neulonets* chosen with a probability based on fitness.
      i. Reproduce an existing *neulonet* by copying it into the new population.
      ii. Create two new *neulonets* from two existing ones by genetically recombining chosen parts using a crossover operation applied at a randomly chosen crossover point within each *neulonet*.
      iii. Create a new *neulonet* from an existing one using a mutation operation at a randomly chosen mutation point.
3. The *neulonet* that is identified to be the best individual is designated as the solution to the problem.

For efficiency, the *neulonet* structure that undergoes evolution is a labeled tree implemented using a prefix-ordered, jump-table approach [Keith and Martin, 1994]. The initial population of *neulonets* are generated in a similar fashion as Koza's [1992] "ramped-half-and-half" generative method. We use a normalized fitness measure $f(\epsilon, \sigma; \kappa)$ based on the errors produced by the *neulonet*, ϵ, as well as the size of the *neulonet*, σ. The factor, $\kappa \in [0, 1]$, is used to weigh the effects of accuracy over size in the fitness measure. A higher value for κ places more emphasis on finding an accurate solution at the expense of the size of the *neulonet*.

The crossover operation involves swapping the chosen parts of two *neulonets* with constraints imposed on the swapping process to preserve the syntactic integrity. For instance, swapping should not leave any dangling intermediate output nodes as such nodes will not be able to receive input values during firing. Figures 5 and 6 illustrate a crossover operation on two *neulonets* before and after the operation.

The mutation operation involves changes to a chosen *neulonet*. A random mutation point is picked such that the *neulonet* whose root is the mutation point is replaced by another randomly generated *neulonet*. Figure 7 shows the effect of the mutation operation on a *neulonet* with the "Conformity" *net rule* replaced by a "Priority" *net rule*.

The probabilities assigned are: (i) reproduction: 15%, (ii) crossover: 80%, and (iii) mutation: 5%.

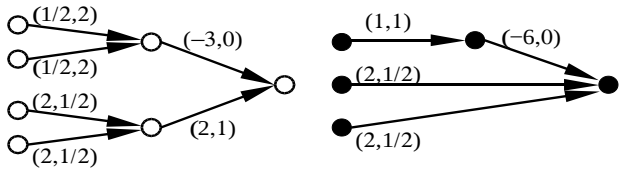

Figure 5: Two *neulonets* before crossover operation.

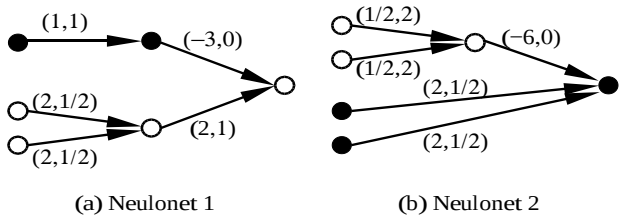

Figure 6: Two *neulonets* after crossover operation.

Note that the "Conformity" *net rule* appears redundant in the activation of a *neulonet* and might be deemed not to be doing anything significant in the evolutionary process. This kind of *net rule* is similar to an *intron* in biology, which is a chromosome that is never expressed and provides spacing between the genes [Angeline, 1994]. However, Levenick [1991] notes that *introns* are useful in genetic algorithms.

4 Proposed Method

In [Tan *et al.*, 1996], it was assumed that the knowledge engineer has some prior knowledge to construct a *neulonet*, which is later subjected to further refinement training. What if the knowledge engineer has only a set of examples without any other knowledge? One approach is to construct a *neulonet* with random weights and train it with the examples. The resultant *neulonet* is no different from an ordinary neural network, as it is difficult to interpret the logic semantics from the seemingly meaningless set of weights. Another approach is to solve a set of inequalities arising from the activation function in equation (1) [Teh, 1995]. The solution is not unique. Moreover, the process becomes more difficult with increasing number of inputs and it may not be always possible to find a set of interpretable weights to satisfy all the inequalities.

The genetic programming paradigm proposed here provides a third alternative for constructing a *neulonet* from examples. We shall illustrate the process using a simple example from the Space Shuttle Landing Domain [Michie, 1988].

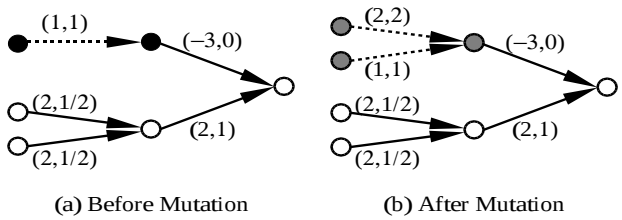

Figure 7: *Neulonet* with a mutated *net rule*.

This data set comprises 15 instances and 6 attributes. Table 8 shows each instance being classified as either to auto-land or not. To conform to the input requirements of the *neulonet* structure, every distinct attribute-value pair has a corresponding boolean attribute in a transformed data set. The valid values for these new attributes can either be yes, no or unknown. In our example, the 6-attribute data set transforms to a 16-attribute data set of boolean values.

Stable	Error	Sign	Wind	Magnitude	Vis	Class
?	?	?	?	?	no	auto
xstab	?	?	?	?	yes	noauto
stab	LX	?	?	?	yes	noauto
stab	XL	?	?	?	yes	noauto
stab	MM	nn	tail	?	yes	noauto
?	?	?	?	OutOfRange	yes	noauto
stab	SS	?	?	Low	yes	auto
stab	SS	?	?	Medium	yes	auto
stab	SS	?	?	Strong	yes	auto
stab	MM	pp	head	Low	yes	auto
stab	MM	pp	head	Medium	yes	auto
stab	MM	pp	tail	Low	yes	auto
stab	MM	pp	tail	Medium	yes	auto
stab	MM	pp	head	Strong	yes	noauto
stab	MM	pp	tail	Strong	yes	auto

Table 8: Space Shuttle Landing Domain data set.

For a particular *neulonet* evolution run, a 100% accurate solution was produced in generation 5 which consisted of 8 basic *net rules*. Further evolution produced another 100% accurate solution in generation 11 as shown in figure 9. This solution consisted of a "Priority" rule rooted at P, a "At-Least-2-Yes" rule rooted at L, and a "Negation" rule rooted at N.

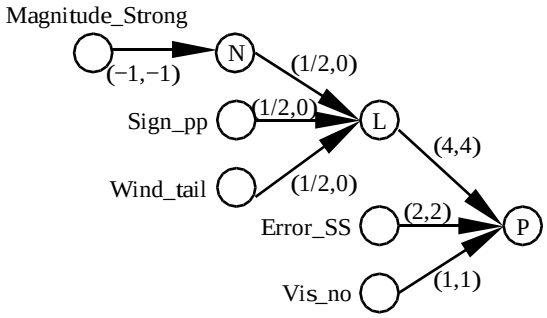

Figure 9: A solution to the Shuttle-Landing classification problem.

5 Extracting Rules from Neulonets

Extracting rules from neural networks has been studied by many researchers [Gallant, 1993; Gaudet, 1996; Setiono and Liu, 1996; Tan, 1997; Towell and Shavlik, 1993]. Existing work, however, basically utilizes standard logic, such as negation, conjunction and disjunction. The rules extracted are generally in the form of a decision tree equivalent to an AND/OR tree based on some classification of the example data. In our present work, the goal is to extract human decision logic from the *neulonets*. The extraction is in fact a straightforward process because the *neulonets* constructed are just a composition of *net rules* which by themselves fully express the human logic in use. As for the case of the Space Shuttle Landing Domain classification problem, the following *net rules* are easily identified from the solution in figure 9.

N : **Negation**(Magnitude_Strong)
L : **At-Least-Two-Yes**(N, Sign_pp, Wind_tail)
P : **Priority**(L, Error_SS, Visibility_no)

In layman terms, the decision to auto-land the space shuttle is biased towards any two or more positive factors for a "pp" sign, tail wind and a magnitude that is not strong. Otherwise, the presence (absence) of an "SS" error will result in a decision to auto-land (manual-land) the shuttle. If this error is unknown, then the decision to auto-land (manual-land) depends on a negative (positive) visibility.

6 Empirical Study and Discussion

Our analysis above shows that the proposed approach of using *net rules* in genetic programming should perform well for data sets which encompass some form of ordered logic reasoning. This analysis will be further confirmed via experiments. In particular, we wish to verify whether using an extended set of *net rules* as logical units is comparable to, if not better, than merely using a limited set of *net rules* comprising the standard boolean logic of negation, conjunction and disjunction (rules (1), (5) and (6) of figure 3). We selected data sets commonly used and publicly available from the UC Irvine data repository [Blake and Merz, 1998]. Data sets containing discrete attribute data types were transformed to an equivalent binary data set with one attribute for each attribute-value pair in the original set. Moreover, data sets containing continuous-valued attributes were pre-processed using a Chi2 Discretizer [Liu and Motoda, 1998] prior to the transformation. For the case of data sets having three or more class values, a separate data set that performs a boolean classification for each class value was created. Table 10 summarizes the data sets used in our experiments.

Data Set	# Instances	# Attributes
Shuttle-Landing-Control	15	16
Iris-setosa	150	123
Iris-versicolour	150	123
Iris-virginica	150	123
Monk-1	432	17
Monk-2	432	17
Monk-3	432	17
Voting-records	435	32
Breast-Cancer-Wisconsin	699	90
Mushroom	8124	125

Table 10: Data set summary. #Attributes denotes the number of boolean attributes in the transformed data set.

The entire library of *net rules* given in figure 3 was used for our experiments. Each of the *net rules* (4) to (13), were represented by their 2- and 3-arity equivalents, while a 4-arity "2-out-of-3 Unless Overridden" rule (14) was used. As a large initial population was required to cater for a wide variety of *neulonets* that could be evolved, we implemented a distributed parallel GP-system on an AP-3000 Fujitsu distributed memory parallel processing system consisting of 32 nodes using a ring-type connection [Niwa and Iba, 1996].

Data Set	*Neulonet*	Standard
Shuttle-landing	100% 3.0 (34.7)	100% 5.5 (48.3)
Iris-setosa	100% 1.0 (18.0)	100% 1.0 (12.0)
Iris-versicolour	99.8% 5.6 (230.0)	99.3% 8.0 (286.7)
Iris-virginica	99.6% 5.3 (154.0)	98.8% 3.0 (133.3)
Monk-1	100% 5.6 (31.6)	100% 4.7 (24.0)
Monk-2	99.8% 19.2 (456.4)	93.2% 20.8 (544.5)
Monk-3	100% 3.0 (26.6)	99.1% 4.0 (30.3)
Voting-records	99.6% 14.3 (356.3)	97.7% 13.0 (538.7)
Breast-cancer	99.5% 20.0 (552.0)	99.4% 20.2 (666.0)
Mushroom	100% 6.5 (46.2)	100% 6.25 (71.3)

Table 11: Experimental results depicting classification accuracy for the best individual. The numbers below the accuracy value denotes the number of decision nodes and (number of generations).

In our experiments, we used an initial population of 10,000, each having a depth of not more than four component *net rules*, and allowed to evolve to a depth of not more than 17. The evolutionary process would proceed until the classification accuracy and size of the fittest individual were unchanged for the last 100 generations. The weighting factor κ used in the fitness measure was set to 0.99. To further simplify the best evolved *neulonet*, a set of eight most commonly used simplification rules were applied recursively to identify component *net rules* for elimination or recombination. Results for evolving the best *neulonet* solution from an average of 10 runs using the set of *net rules* versus using standard boolean logic are presented in Table 11.

6.1 Classification Accuracy

In terms of the classification accuracy, *neulonet* evolution provided a comparable, if not better, result than standard logic evolution in all cases. This was especially true for data sets having an ordered logic reasoning as in the case of the Shuttle Landing Domain problem, or when the classification rules encompassed a "quantification" of standard boolean logic. For example, in the Monk-2 data set, the classification rule is given by [Thrun *et al.*, 1991] as follows:

Exactly two of $a_i = 1 \;\; \forall i \in \{1, 2, 3, 4, 5, 6\}$

Clearly, it would be difficult to achieve a high classification accuracy using only a composition of standard boolean logic units. However, in the case of *neulonet* evolution, the expressive power inherent in the set of *net rules* allowed for a more accurate solution tree to be evolved.

6.2 Solution Size

Due to the *neulonet's* ability to handle more complex decisions, *net rule* evolution provided more compact solutions than their standard boolean counterparts for the same classification accuracy, particularly for data sets having non-trivial classification rules. Using the best evolution runs for both empirical approaches in the Monk-2 data set, the profiles for the number of decision nodes is shown in figure 12 for accuracies between 70% to 100%. Observe that the profiles for both approaches are comparable in the case of classification accuracies of less than 90%, indicating that the classification rule learned thus far was still relatively simple. However, as the required accuracy increased, the apparent power of *neulonets* in deriving complex decision rules resulted in a significantly smaller solution size.

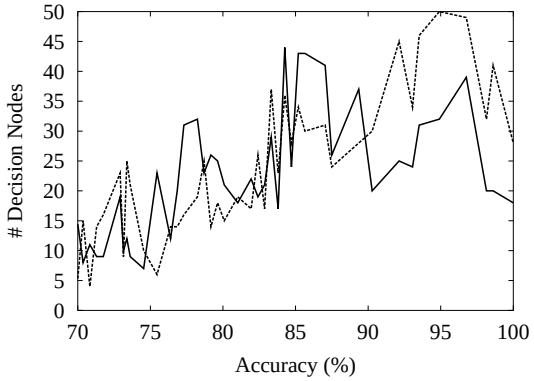

Figure 12: Profile for the number of decision nodes during *net rule*(solid-line) versus standard boolean logic(dotted-line) evolution.

6.3 Number of Generations Required

A general drawback in *net rule* evolution was the longer time needed to converge to an ideal solution due to the larger size of the component *net rule* library. It has to be noted that the experiment conducted actually gave the standard logic an advantage in that the rule set is small (only negation and 2- or 3-arity conjunctions and disjunctions), while the population sizes in both *neulonet* and standard logic evolution approaches were kept the same at 10,000 individuals. As a result, there were more opportunities for the small set of standard logic *rules* to quickly evolve to ideal individuals within the population. Thus for problems involving simpler decisions, standard logic evolution attained the required accuracy faster. However, there were cases in which *neulonet* evolution was faster. This was observed from the accuracy profile for the Monk-2 data set in figure 13.

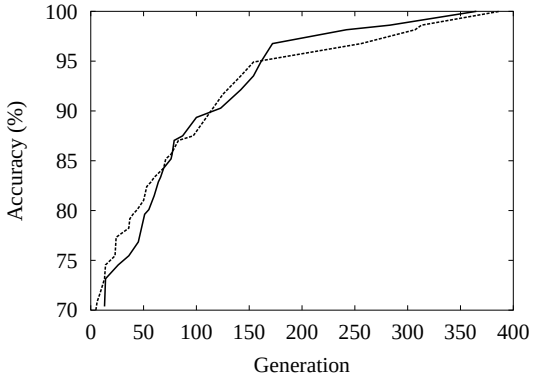

Figure 13: Accuracy profile for *net rule*(solid-line) versus standard boolean logic(dotted-line) evolution.

For accuracies of less than 95%, standard logic evolution required slightly less generations. However, as the demand in accuracy increased, it became increasingly difficult to evolve a solution using only standard logic rules. The added advantage in expressing complex rules during *neulonet* evolution, on the other hand, produced a faster rate of convergence.

7 Conclusion

Genetic programming proves to be an interesting paradigm in constructing *neulonets* with the prescribed human logic *net rules*. The paradigm is also amenable to refinement training. A knowledge engineer could start by constructing *neulonets* based on the human expert's prior knowledge. The constructed *neulonets* may then be subjected to the genetic programming evolutionary process. Thus step 1 of the genetic programming solution (i.e. generation of an initial random population) may be skipped and instead, the process will start from step 2 onwards.

This new mode of neural logic network learning, whether learning from scratch or learning by refining the existing network, preserves the logic semantics understandable for general human decision making. The size of the *net rule* library determines the granularity of the decision steps and the processing time. A large library enhances the capability in expressing nuances among different decisions but at the expense of a longer time to converge.

Further experiments will be carried out to investigate this trade-off issue. Variants of crossover and mutation processes, and fitness measures will also be studied. A long term plan in future is to apply genetic programming on fuzzy neural logic networks [Teh, 1995]. For such networks, the values for input and output ordered pairs are real-valued between 0 and 1. Genetic programming will thus be an attempt to evolve the best fuzzy decision rules. We envision an even more exciting horizon of fuzzy *neulonet* learning with genetic programming.

References

[Angeline, 1994] Peter J. Angeline. Genetic Programming and Emergent Intelligence, in Kinnear, K.E, ed., 1994. *Advances in Genetic Programming*. Cambridge, Mass.: MIT Press, 75–97.

[Armstrong and Thomas, 1996] William W. Armstrong and Monroe M. Thomas. Adaptive Logic Networks. In *The Handbook of Neural Computation*, Fieseler, E. and Beale E., eds., Institute of Physics Publishing and Oxford University Press USA.

[Blake and Merz, 1998] Catherine L. Blake and C. J. Merz. UCI Repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html` Irvine, CA: University of California, Department of Information and Computer Science.

[Gallant, 1993] Stephen I. Gallant. Extracting Rules from Neural Networks. In *Neural Network Learning and Expert Systems*, chapter 17, 315–330.

[Gaudet, 1996] Vincent C. Gaudet. Genetic Programming of Logic-Based Neural Networks. In *Genetic Algorithms for Pattern Recognition*, Pal, S.K. and Wang, P.P., eds., Boca Raton: CRC Press, 213–226.

[Golubski and Feuring, 1999] Wolfgang Golubski and Thomas Feuring. Evolving Neural Network Structures by Means of Genetic Programming. In *Genetic Programming: Proceedings of the Second European Workshop, EuroGP'99*, Ricardo Poli, et al, eds., Springer-Verlag Lecture Notes in Computer Science, Vol. 1598, 211–220.

[Keith and Martin, 1994] Mike J. Keith and Martin C. Martin. Genetic Programming in C++: Implementation Issues, in Kinnear, K.E. ed., 1994. *Advances in Genetic Programming*. Cambridge, Mass.: MIT Press, 285–310.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[Levenick, 1991] James R Levenick. Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Belew, R.K; and Booker, L.B., eds., San Mateo, CA: Morgan Kaufmann Publishers Inc., 123–127.

[Liu and Motoda, 1998] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* The Kluwer International Series in Engineering and Computer Science, Vol. 454, 161–168, Kluwer Academic Publishers, Boston.

[Michie, 1988] Donald Michie. The Fifth Generation's Unbridged Gap. In *The Universal Turing Machine: A Half-Century Survey*, Rolf Herken (ed.), 466–489, Oxford University Press.

[Niwa and Iba, 1996] Tatsuya Niwa and Hitoshi Iba. Distributed Genetic Programming: Empirical Study and Analysis. In *Genetic Programming : Proceedings of the First Annual Conference 1996*, edited by John, R. Koza *et al.*, 339–344. Cambridge, Mass. : MIT Press.

[Setiono and Liu, 1996] Rudy Setiono and Huan Liu. Symbolic Representation of Neural Networks, *Computer*, 29(3), 71–77.

[Tan, 1997] Ah Hwee Tan. Cascade ARTMAP: Integrating Neural Computation and Symbolic Knowledge Processing, IEEE Transactions on Neural Networks, 8(2): 237–250.

[Tan *et al.*, 1996] Chew Lim Tan, Tong Seng Quah, and Hoon Heng Teh. An Artificial Neural Network that Models Human Decision Making, *Computer*, 29(3), 64–70.

[Teh, 1995] Hoon Heng Teh. *Neural Logic Network, A New Class of Neural Networks Called Neural Logic Networks*, Singapore: World Scientific.

[Thrun *et al.*, 1991] Sebastian Thrun, *et al*. The MONK'S Problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU–CS–91–197, Carnegie Mellon University, Pittsburgh, PA.

[Towell and Shavlik, 1993] Geoffrey G. Towell and Jude W. Shavlik. Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning*, 13(1), 71–101.