Towards a High-Level Hybrid System of Neural Classifiers

Napoleon H. Reyes
Physics Department
De La Salle University
2401 Taft Avenue, Manila, Philippines
Tel: +63 2 536 0229
cosnhr@mail.dlsu.edu.ph

Dr. Arnulfo P. Azcarraga
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
Tel: +65 874 2727 Fax: +65 779 4580
dcsapa@nus.edu.sg

ABSTRACT

Neural Networks and Expert Systems are two complementary approaches to building intelligent systems. Neural Networks lend themselves naturally to learning, but can be quite obscure as to their basis for the mapping they make of input patterns and output responses. Expert Systems, on the other hand, are particularly attractive in terms of being able to explain the basis for the diagnosis. This type of complementarity gives rise to hybrid systems that combine Expert Systems and Neural Networks, and such a system is described here. This system transforms a rule-based Expert System into a multi-layered Neural Network, then refines the system through training using live data. A popular benchmarking problem for hybrid systems - the DNA promoter recognition problem [Fu, personal communication], serves to demonstrate the inner workings of the proposed hybrid system.

Keywords

Expert Systems, Neural Networks, Hybrid Systems, Explanation-based Systems, Empirical Learning Systems.

1. INTRODUCTION

With the advent of Neural Networks and Expert Systems, the computing power of information processing systems catapulted to a new dimension, unraveling the complexities of seemingly unsolvable computer problems in a variety of scientific and engineering disciplines. Despite all the breakthroughs in the intelligent systems community, there is no one technique that is suitable for all types of problems. Every intelligent scheme is compelled with computational properties, like the ability to extract patterns from a set of information, learning rate, and the ability to explain the intricacies behind decisions. Taking into account two of the most popular intelligent techniques, namely Neural Networks and Expert Systems, it has been shown that both have the capability of modeling complex nonlinear processes to arbitrary degrees of accuracy. However, while Neural Networks are exceptionally good at recognizing patterns without prior knowledge about the rules, such systems do not provide details on how decisions are made. Expert Systems on the other hand, can reason with imprecise information, but cannot discern knowledge as effectively. It is in the light of these limitations that an intelligent hybrid expert-neural system is developed.

Related research, such as the KBANN system [12,13,14] presents an algorithm for the transformation of rules to Neural Network and fine-tuning of the Network, but leaves the mapping of the rules by manual inspection. This paper describes a hybrid system that utilizes and enhances the techniques described in the KBANN system and TreeNet [1]. It features an automated Rules-to-Neural Network mapping technique that generates a Network equivalent to that done by manual inspection, and employs a modified Backpropagation learning scheme [15] for fine-tuning the system at an improved convergence speed.

As an overview of the system (see Figure 1), a set of rules from an Expert System is transformed into its Neural Network equivalent (Transformed Model) by setting the network topology, initial weights, and biases in accordance to the interdependencies among the rules. Then, by subjecting the system to live data, the transformed model is refined.

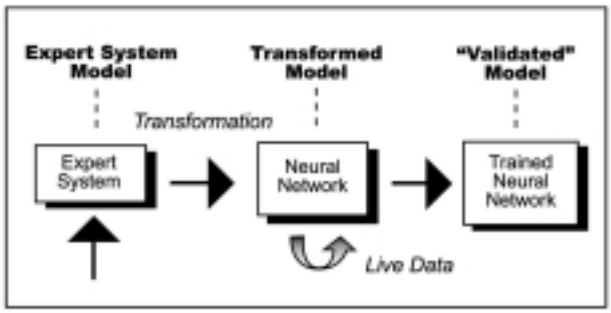

Figure 1. Overview of the Hybrid System.

2. HYBRID SYSTEM DESIGN

Generally, the idea is to devise a methodology for constructing a Neural Network that adheres to the precedence and association of rules in decision-making. By traversing a hierarchy of supporting facts and intermediate conclusions (see Figure 2), the final decision is arrived at. This structure conforms to our natural perception of things in life, which is extremely vital in shedding some light to the inner workings of the network.

The proposed hybrid architecture embodies the chain of rules as chain of nodes, where the supporting facts are represented as input units (leaf nodes), intermediate conclusions as hidden units and final conclusions as output nodes [11]. In addition, the weighted links and bias units accurately quantify the interdependencies among the rules.

Many factors in the design process could severely degrade the system's efficiency. Primarily, there are four phases of system design, namely knowledge representation, rules-to-network

conversion, input features classification, and knowledge refinement. In the subsequent sections, each design phase will be expounded, along with a discussion of problem issues that deter system performance, and their corresponding resolution.

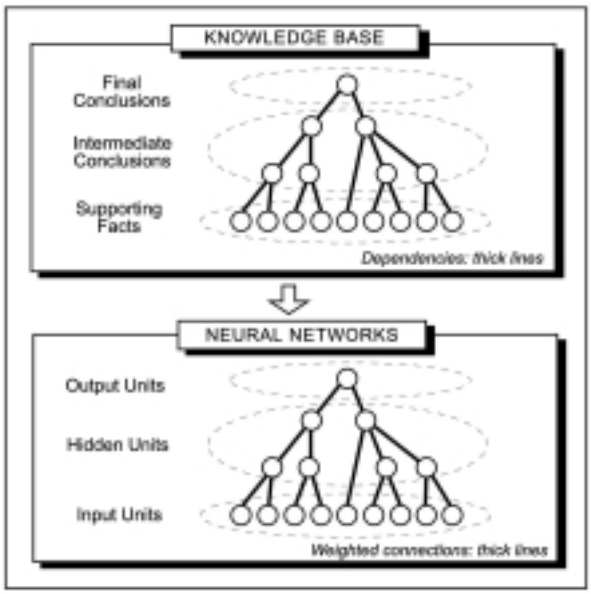

Figure 2. Correspondences between a rule set and a Neural Network.

2.1 Knowledge Representation

When prior domain theory is available, it is essential that the rules be verified of discrepancies first before presenting to the system for network translation. The object of verification is to remove undesired rule structures [16] that inhibit network conversion (see Table 1).

2.2 Rules-to-Neural Network Conversion

It is of primary importance that the Neural Network be constructed in adherence to the interdependencies among the rules (see Figure 2). The hierarchical structure is essential in keeping track of the dynamic synthesis of thoughts, as the signals propagate from the input units (supporting facts) to subsequent hidden units (intermediate conclusions), until it converges in the output unit (final decision).

Basically, the translation is accomplished by mapping each rule to its corresponding Neural Network component. By building the network this way, significant features that are equivalently worth thousands of training time are immediately injected into the network. Consequently, problems that usually arise during training are substantially reduced (e.g. spurious correlations between data, irrelevant features, etc.).

As a prerequisite to a successful translation, the rules must be conjunctive, non-recursive, variable-free, and free from undesired rules structures (see Table 1). Furthermore, the rules must combine in a hierarchical fashion so as to permit the creation of derived features. Therefore, in the rule set, some rules do not map directly from inputs to outputs, but rather to some intermediary conclusion. An abstract specification of the rules-to-network translation is presented in the following subsection (see Table 2), followed by a detailed discussion of each step and a real-world example.

Table 1. Undesired rule structures

1. Redundant Rules
   Two rules share the same antecedent and consequent.
2. Conflicting Rules
   The antecedents of two rules are equivalent, but one or more of their conclusions are contradictory.
   For example,
   Rule 1: If A, Then X.
   Rule 2: If A, then not X.
3. Subsumed Rules
   Two rules have the same conclusions, but one contains fewer conditions than the other in the if-part.
   For example,
   Rule 3: If B and C, then Y.
   Rule 4: If B, then Y.
   Rule 3 is subsumed by rule 4; whenever the former succeeds, the latter also does but not vice versa.
4. Unnecessary If Conditions
   For example,
   Rule 5: If D and E, then Z.
   Rule 6: If D and not E, then Z.
   → The conditions "E" and "not E" are unnecessary.
5. Circular Rule Chain
   The chaining of rules forms a cycle. For example,
   Rule 7: If P, then Q.
   Rule 8: If Q, then R.
   Rule 9: If R, then P.
   → In this case, the system will enter an infinite loop at run time, unless it has a special way of handling this situation.

Step 1. *Rewriting the rules*. Assuming that the rules already comply with the imposed restrictions in Section 2.1, the first step transforms the rules in a form that can be recognized by the system. This involves scanning every element of the rule, adding reserved words and symbols, and rewriting the rules with multiple consequents as separate rules (see Table 3 for the syntax). The hybrid system admits rules with multiple antecedents, but strictly with only one consequent per rule.

Step 2. *Postfix conversion*. In the same way mathematical expressions are translated into their corresponding Postfix form [5], expert rules are broken down into multiple components and converted into Postfix form one element at a time.

Table 2. Rules-to-Neural Network algorithm

1. Rewrite the rules describing the system in condition-action pairs (if..then form) so that there is only one consequent per rule.
2. Convert each rule to its Postfix form.
3. Map each rule (in Postfix) into a Neural Network (sub tree / sub network).
4. Accumulate all sub trees together into one big net, then optimize.
5. Label all nodes according to their respective layers, operator and links.
6. Add links not specified by the translation between subsequent nodes, provided that leaf nodes are not connected to other nodes belonging to lower layers.
7. Initialize all weighted links.
8. Compute biases.

Table 3. Rule-writing syntax

Rule No.

IF [not] variable1 = value1 [and/or variable2 = value2]
THEN action = task1.

Restrictions:

1. A rule must end with a period.
2. Identifiers must not be any of the following reserved words and symbols:
   Reserved words: IF, THEN, And, Or, Not
   Reserved symbols: =, (,), .
3. Antecedent and consequent identifiers must be at least two characters long. (e.g. temp, pressure, time, speed, tastiness, main-component)
4. Antecedents are preceded by the word "if".
5. Consequents are preceded by the word "then".
6. Multiple antecedents can be linked together by using a pair of parentheses and operators like "and" and "or".
7. Precedence of operators can be controlled by nested parentheses.
8. Negated antecedents must be preceded by the word "not".
9. Use an equal sign to associate a value with its corresponding antecedent or consequent.

Step 3. *Mapping*. In a similar fashion Postfix expressions are evaluated for mathematical purposes, the Neural Network is constructed according to the sequence of elements in the Postfix stack. Throughout the mapping, the precedence of operators is preserved and redundant operators removed, thereby reducing unnecessary nodes that could deter network performance later on.

Postfix expression = 1 2 3 or and 4 5 or and.

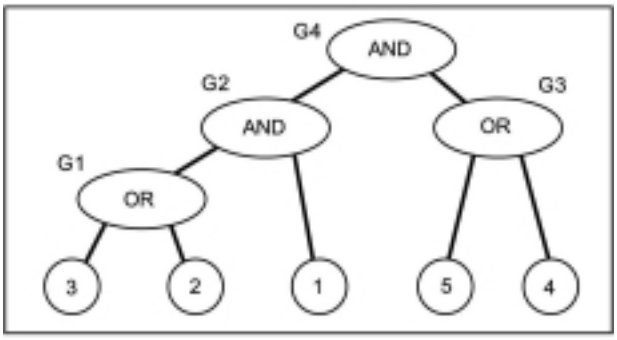

Figure 3. Initial Neural Network (Subtree)

The initial Network created from the Postfix expression after mapping. This network however, still contains redundant nodes, labeled G2 and G4.

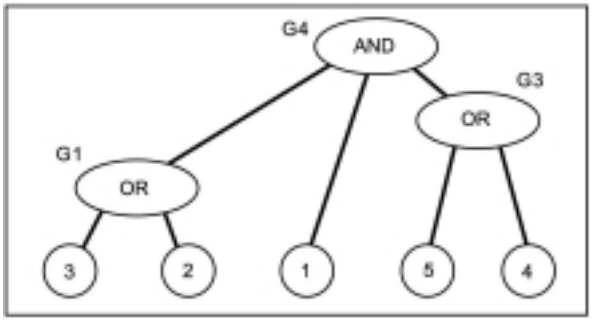

Figure 4. Simplified Neural Network (Subtree)

The Network is then simplified by combining two consecutive similar operators together as one operator.

Step 4. *Accumulating subtrees and optimizing the Big Net.* In this step, all subtrees representing each rule are accumulated into the big net one at a time. Throughout the integration process, duplicate nodes are checked and subtrees are joined together in agreement with their interdependencies. Merging two nodes that posses "OR" operators as a single node optimizes the big net, getting rid of redundant nodes.

Step 5. *Labeling nodes.* In this step, each unit in the big net is labeled according to their respective layers. Starting from the root node as layer one, until the leaf nodes are reached, the layer numbering is done in increasing number. This labeling is a necessary precursor in determining the wiring between nodes, and bias computations in the succeeding steps.

As can be seen in Figure 5, each node is labeled according to its minimum distance from the root node. The labeling is done in increasing order, starting from the top, moving downwards, and in this figure, the maximum layer is 3.

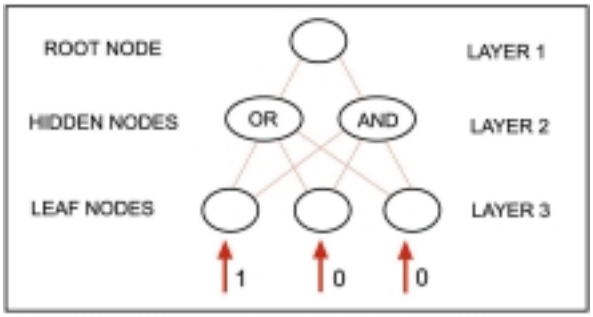

Figure 5. Labeling nodes

Step 6. *Adding links.* In this step, links not defined by the rules are added to account for any missing information that could have been possibly left out during the writing of the rules. This task involves fully connecting each of the nodes to the rest of the nodes in the subsequent layers (both upper and lower layers). Consequently, by referring to the layer number assigned in Step 5, the connections are easily made, provided that leaf nodes are not connected to other nodes residing at lower layers.

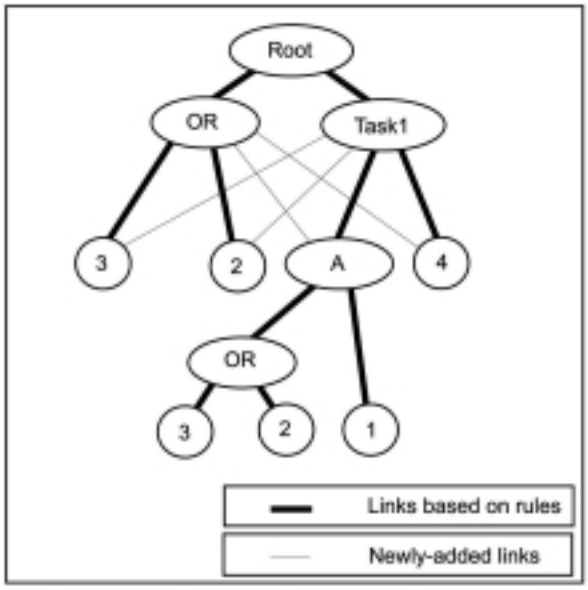

Figure 6. Adding links not defined by the rule set.

Subsequent nodes are fully connected, provided that leaf nodes are not connected to other nodes below.

Step 7. *Initializing weights*. In this step, all links defined by the rules are initialized with a weight of value 3.0 if it is not negated, and -3.0 if negated otherwise. On the other hand, newly added links defined in Step 6 are assigned a weight of zero. After all weights have been accounted for, the weights are perturbed with random values very near zero (e.g. 0.012, 0.025, etc.) so as to avoid training problems that occur in symmetrical networks [12].

Step 8. *Computing biases*. In order to employ the interrelationship among the set of rules, upon which the Neural Network is based, the biases are computed in congruence with the logical operators used and the number of mandatory antecedents. The succeeding bias formulas were derived based on the initial weights set, and target activation. When the rules satisfy the conditions, the activation is approximately 0.91; otherwise, 0.1.

Table 4. Bias formulas

1. For handling disjuncts (OR) Bias = $\omega - \varphi$ where $\omega = 3.0$ weight, and $\varphi = 2.3$
2. For handling conjuncts (AND) Bias = $n\omega - \varphi$ where n = number of positive mandatory antecedents $\omega = 3.0$ and $\varphi = 2.3$
3. For handling certainty factors (cf) - still untested Bias = $-ln((100 - cf) / cf)$ where cf = certainty factor range [0-100] (e.g. 25, 80, 90)

2.3 Input/Output Features Classification

Once the entire network topology, weights and biases have been established based on the domain theory, the network is prepared for training. The preparation involves explicitly defining the input features that associate every input unit with their corresponding exemplars, and generating a table of exemplars that takes into account all possible values for each input unit.

By inspecting each and every leaf nodes in the Neural Network and recording their unique values, all the possible input values are accounted for. These input values are then assigned a specific data type that is used for encoding. On the other hand, all child nodes of the root node are extracted to form the output nodes where the final judgments are made.

In the constructed Neural Network, all leaf nodes correspond to the global input units of the system. Exemplars are fed into their corresponding input nodes as indicated by the input labels, and input signals are decoded with respect to their specific input feature types.

2.4 Knowledge Refinement

The proposed system refines the network by utilizing a modified Backpropagation algorithm [15]. Since all bias values are computed based on the initial domain theory, these values are kept constant during training. The training paradigm uses a different error formula, so that large error signals are propagated through the network when the difference between the actual and desired output is large. This error signal formula is based on a different total error-of-performance function that was described by Van Ooyen in his paper [15]. However, Van Ooyen did not assign any name for identifying such an error function for citation.

Nevertheless, Towell mentioned in his dissertation that Rumelhart suggests a similar error signal formula based on a function called normalized exponential error function (NEEF), and so this paper refers to the error function as NEEF [12].

To be able to gauge the performance of the network relative to existing learning schemes, the performance of the hybrid system is compared against Standard Backpropagation and the KBANN training algorithm, which is based on the cross-entropy error function. Table 5 provides a list of the error signal formulas used for comparison, and these are based on the different total error-of-performance function that is to be minimized by the learning algorithm.

Network refinement is accomplished by presenting all instances from the table of exemplars. To find out which configuration leads to optimum Network performance, different network parameters were set. Among the network parameters that were varied during training are learning rate, training scheme, epoch, and a multitude of rule sets suitable for defining the same problem.

Table 5. Error signal formulas

1. Standard Backpropagation ei = [di - ai] * [(1 - ai) * ai]
2. Cross entropy error function – used by KBANN ei = log2 * [(di / ai) – ((1 - di)/(1 - ai))] * [(1 - ai) * ai]
3. Normalized Exponential Error Function (NEEF) ei = [ai - di]
*ei - error signal propagating back from each output unit

3. RESULTS AND ANALYSIS

There are various ways by which a hybrid system can be configured, the succeeding subsections discusses how well the hybrid system responded when subjected to different rule bases, and learning techniques. The empirical results quantify the extent to which these different aspects of design affect system performance.

The hybrid system is tested in solving two different applications, namely the XOR problem, and the DNA promoter recognition problem. In the subsequent sections, the results for each of these problems are fully discussed.

3.1 Utilizing Prior Domain Theory

To investigate how prior domain theory affects the convergence speed of Neural Networks during training, the hybrid system is tested in solving the DNA promoter recognition problem in twelve different configurations. There are three error functions, namely STD BACKPROP, NEEF, and CROSS-ENTROPY; and the other hand, four different initial Neural Network weight and bias settings. All possible permutations (see Table 6) arising from these parameters are tested in solving the DNA problem using the same Neural Network topology, links, number of nodes, and training time.

3.1.1 DNA Promoter Recognition Problem

This problem determines if a DNA nucleotide string is either a promoter or not. Biological literature states that promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. Simply put, it inspects specific locations in the DNA strand where some patterns appear, and classifies the strand according to some conditions.

The rules and data used are obtained from a publicly available database repository for machine learning [8]. After applying the Neural Network translation algorithm, the simplified neural network configuration that was arrived at have the following features:

Number of Layers = 4

Input Nodes = 57

Hidden Nodes = 16

Output Node = 1

3.1.2 Empirical Results

After configuring the system using the parameters shown in Table 6, and then training the system for 100 epochs using 106 exemplars extracted from the repository, it is found that utilizing prior knowledge and the NEEF error function, gained the most desirable result (Network initialized with computed weights and biases, and trained, using NEEF; sum of squared errors = 0.06).

Table 6. Summary of sum of squared errors after 100 epochs

	Error Function		
Initial Network Configuration	**STD. BP**	**CROSS-ENTROPY**	**NEEF**
Zero Weights & Biases	27.74	32.28	0.34
Zero Weights, Computed Biases	13.15	10.06	9.94
Computed Weights, Zero Biases	28.91	37.46	41.43
Computed Weights & Biases	1.65	0.11	0.06

These tabulated results can also be graphically viewed as charts, to observe how the Network converges closer to a solution after each epoch during training (see Figures 7,8 and 9).

Neural Network training time can be extended further than what is shown in the charts to gain lower sum of squared errors; however,

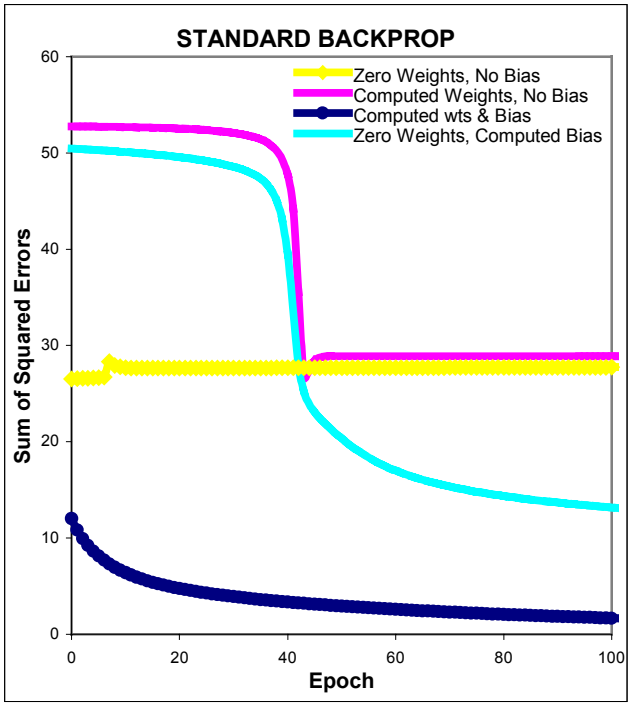

Figure 7. Effects of utilizing prior knowledge on Neural Networks using Standard Backpropagation.

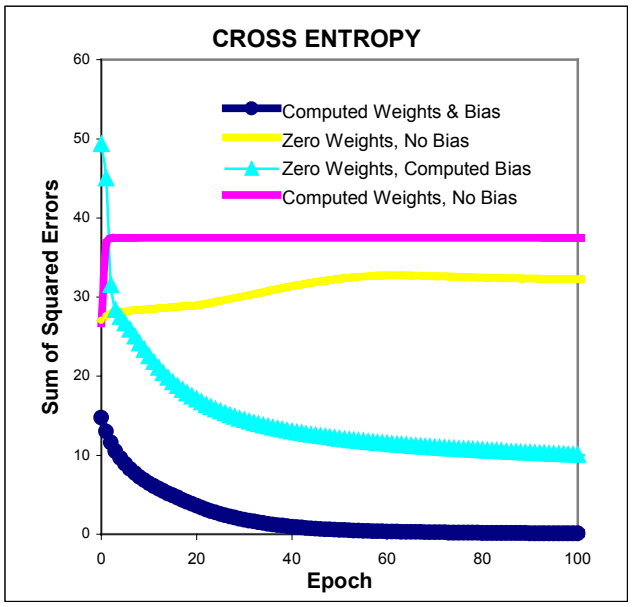

Figure 8. Effects of utilizing prior knowledge on Neural Networks trained using Cross-Entropy error formula.

since convergence speed is the main focus here, the minimum training time was used for comparison (epoch = 100 was empirically found).

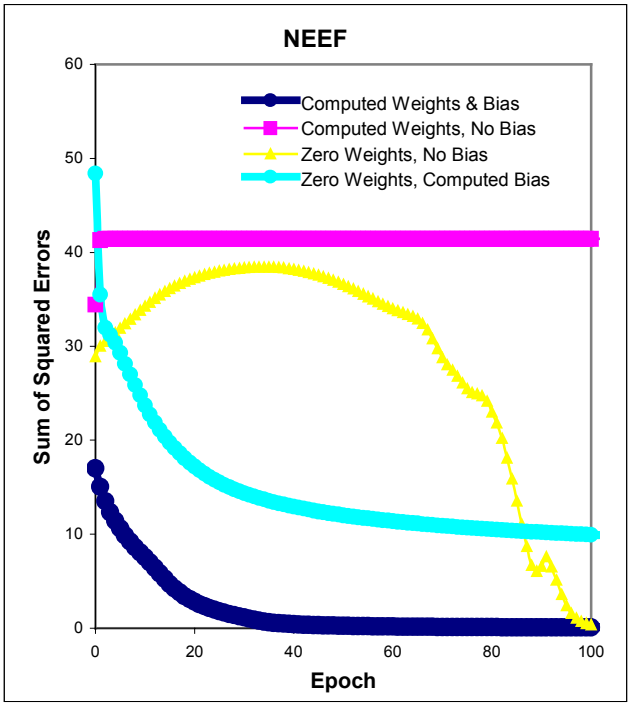

Figure 9. Effects of utilizing prior knowledge on Neural Networks using NEEF error formula.

From Figures 7, 8, and 9, it is evident that Networks that started with computed weights and biases consistently exhibited relatively low sum of squared errors than the rest of the other settings, regardless of what error signal formula is used.

Having this observation in mind, the three training paradigms are then compared which among them notches the fastest convergence when prior knowledge is utilized by the system (see Figure 10)

3.2 Training Paradigms

Based on the observations made in Section 3.1, the Hybrid System is then constructed with computed weights and biases, then trained using the three different training paradigms (described in Section 2.3), namely NEEF, Cross-Entropy, and Standard Back Propagation. The system is then applied in solving the DNA promoter recognition problem and the classic XOR problem (see Figures 10 and 11).

The graphs in Figure 10 show that NEEF notched the fastest convergence, with an error of 0.059 after 100 epochs. This is slightly faster than CROSS-ENTROPY which gained an error of 0.113 after 100 epochs. Lastly, Standard Back Propagation poorly registered an error of 1.655 after 100 training epochs.

From the results in Figure 11, it can be observed that the proposed system (NEEF) registered an error of 0.007 after 695 epochs. On the other hand, CROSS-ENTROPY performed slower than NEEF and gained an error of 0.01 after 1,005 epochs. Lastly, Standard Back Propagation poorly registered an error of 0.1, after 1,505 training epochs.

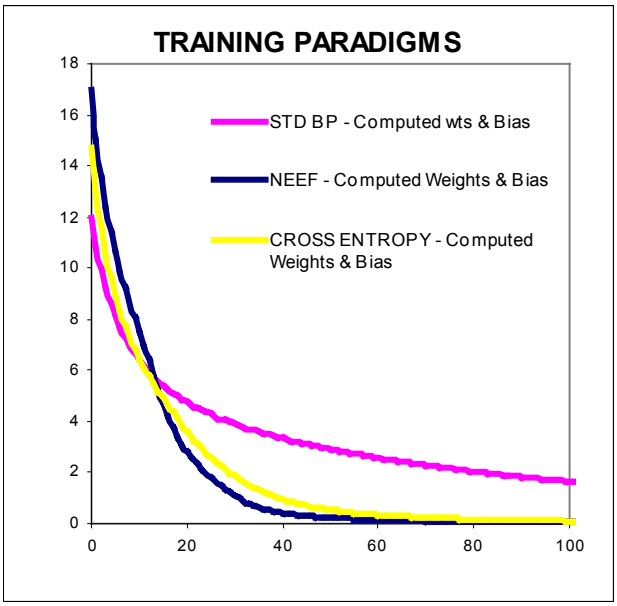

Figure 10. Comparison of the training paradigms when prior knowledge is utilized. (DNA Problem)

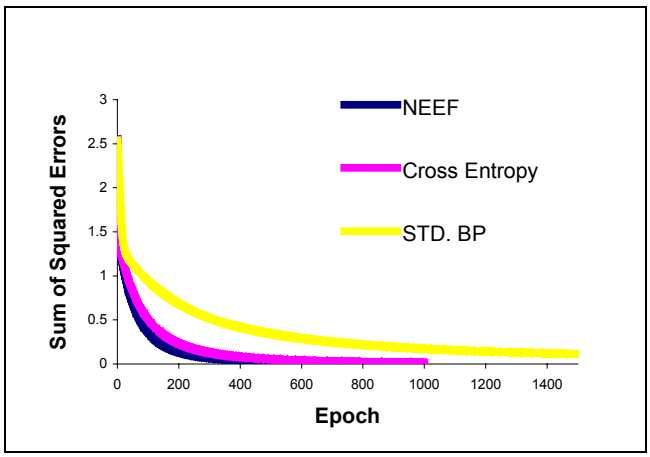

Figure 11. Comparison of training paradigms (XOR).

4. CONCLUSIONS

This research is able to attest to the central thesis that an intelligent hybrid learning system must be able to draw information from prior knowledge about the problem, and training examples. In particular, this work is able to shed some light in addressing the problem of merging two intelligent techniques, namely Expert Systems and Neural Networks. Along the development of the hybrid system, this work has touched on several aspects that renders advantageous to machine learning in general, among which are:

Rules-to-Network Translation

There is no existing literature that provides detailed information as to how exactly rules are mapped to its corresponding nodes automatically. This paper describes an algorithm for automatically mapping rules in a systematic way, and even allows the reduction of nodes, thereby decreasing network-training time.

Knowledge Refinement

By utilizing the formulas presented in a paper by Van Ooyen, the system has exhibited an improvement in training speed, over KBANN's training algorithm and Standard Backpropagation training when the DNA problem was solved.

5. RELATED AND FUTURE WORK

Rule Extraction

This work has successfully accomplished the merging of two complementary methods of learning through a transformation algorithm (Table 2). Different sets of expert rules are translated into their corresponding Neural Networks and further refined using live data. However, the new refined knowledge is yet to be discovered from the trained Neural Network. In line with this study, the next step is to ponder on the extraction of rules from the trained network to extend the system's potential explanation capability. This is yet another work that proves to be worthy of attention especially in the field of medicine where motivations behind decisions are extremely important.

Benchmarking

The hybrid system has outperformed conventional neural networks (Section 3) that start with random valued weights and biases, and is even slightly faster than the KBANN training algorithm [14] in solving the DNA problem. In another perspective, the system should also be tested as to how well it performs when subjected to data that are not seen during training. Most of the tests conducted in measuring the performance use mathematical quantities like Sum of Squared Errors, Maximum Absolute Errors, and Mean Absolute Errors. These formulas describe thoroughly how well the system learns during training; however, the effects of utilizing prior domain theory could be characterized even more by using the leave-one-out testing scheme. Due to time constraints, the DNA Problem was not tested using this scheme because it requires N-1 testing times, where N is the number of all possible instances. In this case, N = 106, and the network converges after 300 epochs. All in all, 31,500 training times are required, with 106 network tryouts required for every trained network.

6. REFERENCES

[1] Azcarraga, A. P., "A High-Level Network of Neural Classifiers", *6th DLSU Computer Conference* Dec. 2 and 3, Philippines, 5-1, 5-13. (1994).

[2] Boz, O., "Knowledge Integration and Rule Extraction in Neural Networks.", EECS Department, Lehigh University. (1995).

[3] Fu, L., *Neural Networks in Computer Intelligence*, Singapore: McGraw-Hill, Inc, pp. 115-142, 275-304. (1994).

[4] Hopgood, A. A., *Knowledge-based systems for engineers and scientists*. Boca Raton, Florida: CRC Press.(1993).

[5] Horowitz, E., S. Sahni, *Fundamentals of Data Structures in Pascal, 2nd ed.*, Computer Science Press, Inc., 82-85. (1987).

[6] Kidd, A. L., *Knowledge Acquisition for Expert Systems*. New York: Plenum Press. (1987).

[7] Levine, R.I., D. E. Drang, B. Edelson, *AI and Expert Systems: a comprehensive guide to Turbo Pascal 2nd edition*, New York: McGraw-Hill, Inc. (1990).

[8] Murphy, P., "Promoters Domain", *UCI Repository of Machine Learning Databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html], University of California, Department of Information and Computer Science, Irvine, California. (1994).

[9] Opitz, D.W., and J.W. Shavlik, "Dynamically Adding Symbolically Meaningful Nodes to Knowledge-Based Neural Networks.", Department of Computer Science, University of Wisconsin, UW TR 1246. (1994).

[10] Posey, C., A. Kandel, G. Langholz, "Fuzzy Hybrid Systems", Kandel A., Langholz G. (eds.), *Hybrid Architectures for Intelligent Systems*, CRC Press, Inc., 173-197. (1992).

[11] Shavlik, J.W., "A Framework For Combining Symbolic and Neural Learning.", Artificial Intelligence and Neural Networks, Steps towards principled integration, edited by Vasant Honavar, Leonard Uhr, Academic Press. (1994).

[12] Towell, G.G., "Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction." Doctoral dissertation, Madison, WI: University of Wisconsin, Computer Sciences Department. (1992).

[13] Towell, G.G., and J.W. Shavlik, "Extracting Refined Rules From Knowledge-Based Neural Networks." , Machine Learning. (1993).

[14] Towell, G.G., and J.W. Shavlik, "Knowledge-Based Artificial Neural Networks." *Artificial Intelligence*, vol. 69 or 70. (1994).

[15] Van Ooyen, A., B. Nienhuis, "Improving the Convergence of Back-Propagation Algorithm", *Neural Networks*, vol 5, 465-471. (1992).

[16] Yang, Q., "Rule Combining - A Neural Network Approach to Design Optimization.", Kandel A., Langholz G. (eds.), *Hybrid Architectures for Intelligent Systems*, CRC Press, Inc., 256-275. (1992).