Proceedings of the ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2007
September 4-7, 2007, Las Vegas, Nevada, USA

DETC07/DAC-34588

A GRAPH GRAMMAR APPROACH TO GENERATE NEURAL NETWORK TOPOLOGIES.

Chaitanya Vempati
Department of Mechanical Engineering
The University of Texas at Austin
Austin, TX 78712
vempati@mail.utexas.edu

Matthew I. Campbell
Department of Mechanical Engineering
The University of Texas at Austin
Austin, TX 78712
mc1@mail.utexas.edu

ABSTRACT

Neural networks are increasingly becoming a useful and popular choice for process modeling. The success of neural networks in effectively modeling a certain problem depends on the topology of the neural network. Generating topologies manually relies on previous neural network experience and is tedious and difficult. Hence there is a rising need for a method that generates neural network topologies for different problems automatically. Current methods such as growing, pruning and using genetic algorithms for this task are very complicated and do not explore all the possible topologies. This paper presents a novel method of automatically generating neural networks using a graph grammar. The approach involves representing the neural network as a graph and defining graph transformation rules to generate the topologies. The approach is simple, efficient and has the ability to create topologies of varying complexity. Two example problems are presented to demonstrate the power of our approach.

1 INTRODUCTION

Neural networks have received tremendous interest among engineers for various applications over the last decade.

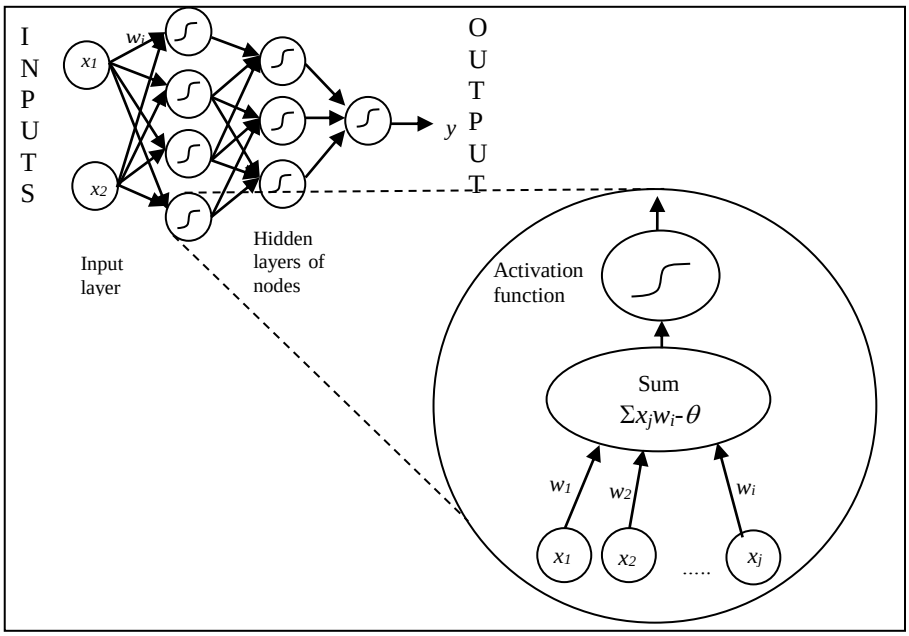


Figure 1: A typical neural network with two inputs and one output and with two hidden layers. The inset shows the processing at each node in more detail.

They are an attractive tool for data analysis because they are flexible, accurate and computationally cheap process modelers [1, 2]. A model usually consists of estimating a quantitative relationship, usually a function, between a set of known variables or inputs and a set of responses or outputs. A neural network is a graph composed of many simple interconnected processing elements (neurons or nodes) operating in parallel. The most common type of artificial neural network consists of three groups of nodes: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units as shown in Figure 1. The topology of a neural network refers to the number of nodes and their organization. These nodes can be interconnected to each other in various parallel and series combinations. The overall function between inputs and outputs is determined by network architecture or topology, the weight of each connection between the nodes and the processing performed within the nodes.

Each node performs a transfer function of the form

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right) \quad (1)$$

Where y = output of the node i.
xij = j'th input to the node,
wij = connection weight between nodes and j,
and θi = bias, a constant added to each node.

f is a nonlinear function, such as a Heaviside, sigmoid, or Gaussian function and is typically known as the activation function. Training a neural network involves modifying the weights and biases of the nodes and connections such that the network best fits the given data. The success of a neural network in modeling a particular problem depends upon this topology, the choice of the processing function within the nodes and the training algorithm. Simpler networks, networks with only a few hidden nodes or sparse connectivity between the nodes, do not have enough flexibility to properly capture complex input-output relationships. This phenomenon is commonly known as underfitting [3] [4]. On the other hand a very large and complicated neural network would not only require very large training times but would also inadvertently fit the noise in the data. Such networks incorrectly map data that has yet to be seen by the network even though the data follows obvious trends. This is called overfitting [3] [4]. Hence it is absolutely critical for a particular problem to have the right topology since the topology of the network puts a-priori limits on complexity. Despite the significant research activity in this area during last few decades, the design of topologies of neural networks for specific applications is still a trial and error process, relying on experts in the field with previous experience in the use of neural networks. But as the complexity of problem to be modeled increases, this method of trial and error becomes difficult and unmanageable. A lot of effort has been focused on developing an elegant method for automatically generating neural network topologies.

This paper presents a novel concept of using graph grammars to generate neural network topologies. We claim that the graph grammar based representation of neural networks is a flexible, robust and an intuitive approach to generating successful neural network topologies. Each topology generated by graph grammar is trained using genetic algorithms and the performance of each topology during training is used to further evolve the topologies. While the methodology and the results presented in this paper are still very new, they are promising and have the potential to be an automated and intelligent process modeler.

2 RELATED WORK

Growing, pruning and genetic algorithm approaches have been widely accepted methods of generating neural network topologies. Growing neural networks involves starting off from a very basic or simple neural network and then adding nodes or connections to generate a neural network that yields a satisfactory data approximation. Lucas [9] describes a method that generates a neural network topology using graph grammars. The emphasis of that method is on understanding the utility of graph grammars to generate a network without the need for training. While the method is promising no results were presented in the paper.

Falhman et al. [5] describe a cascade correlation algorithm which begins with a minimal network, then automatically trains and adds new hidden nodes one by one, creating a multi-layer structure. This algorithm has the advantage of faster training and easy implementation. Pruning of neural networks, as opposed to growing, involves starting off with a large and a very complicated neural network and then removing nodes and connections until a network with satisfactory performance is obtained. However, all possible neural networks are not explored using both of these methods.

Sensitivity analysis [1] is a method in which the effect of removing certain elements (nodes or arcs) of the network is analyzed. The elements that have the least affect are removed first. In general, the sensitivity methods modify a trained network what this means is that the network is trained, sensitivities are estimated, and then connections or nodes are removed.

Slawomir [1] uses the stochastic optimization techniques like genetic algorithms and simulated annealing to reduce the size of the neural network without compromising its generalization capability. There have been interactive approaches and some approaches involving genetic algorithms to prune the neural networks [6]. Reed [7] describes a comprehensive study on such pruning algorithms. He highlights the utility of pruning in reducing the effect of overfitting.

Genetic and evolutionary methods are very popular ways to generate neural network topologies [8, 10, 11]. These algorithms are distinguished by their reliance on a population of search space positions, rather than a single position, to locate extrema of a function defined over the search space. During one search cycle, or generation, the members of the population are

ranked according to a fitness function, and those with higher fitness are probabilistically selected to become parents in the next generation. New population members, called offspring, are created using specialized reproduction heuristics [12]. Generation of neural networks using genetic algorithms (GAs) involves encoding of the network topology. Different parameters have to be defined that capture the structure and connectivity information of the topology. These parameters are then interpreted as a fixed length binary string. As GAs rely mostly on crossover as the main heuristic for reproduction, this can lead to degenerate cases with unconnected nodes or connections [1]. These degenerate cases have to be dealt with during training and further evolution.

Angeline et al. [12] describe a GeNeralized Acquisition of Recurrent Links, as an evolutionary algorithm that nonmonotonically constructs recurrent networks to solve a given task. The algorithm involves the exclusive use of mutation as the reproduction mechanism. This paper also advocates that a genetic algorithm is an inappropriate technique for network topology generation since crossover mechanism they employ is inefficient for topology generation. Hence methods involving genetic algorithms tend to restrict the search space.

Stanley et al. [13, 14] present a method that uses a binary representation of the neural network. The key feature in their method is the use of an innovation number which is a marker that keeps track of the original ancestor of the gene. This addresses some problems involved in the use of genetic algorithms for representing and evolving neural networks.

Yao [4] describes the EPNet system that is built upon an evolutionary programming algorithm which adopts a rank based selection scheme and five mutations; hybrid training, node deletion, connection deletion, connection addition and node addition. Hybrid training is the only mutation in EPNet which modifies neural network weights. The other four grow or prune the network.

The above mentioned algorithms, while addressing the problem of topology generation, make underlying simplifications or generate neural networks in a highly constrained manner [4, 7, 12]. In most methods discussed above, once a topology has been found to be inefficient it is discarded and a new one is generated in its place thus making it almost impossible to regenerate the old topology. The methods also seem to explore only a subset of all the possible neural networks. Generally these methods only define structural modifications that are either very random or overly restrictive. Furthermore, such forms of topology modifications are susceptible to becoming trapped in local minima as they do not conduct an exhaustive search of the entire space [12]. The constraints for neural network generation should be defined by the problem to be solved rather than be inherent in the algorithm that generates the neural networks.

The problem of designing the appropriate neural network can be viewed as two distinct challenges – the first is to develop an intuitive and robust way to generate or design all possible neural network topologies and the second is to quickly predict the worth of a particular design and find the most appropriate design. We present a method that addresses both these challenges. The following sections discuss our approach in detail.

3 NEURAL NETWORK TOPOLOGY DESIGN: OUR METHOD

In this paper the authors present a three step approach of generation, evaluation and guidance for finding the best neural network topology for a particular problem. Generation is the first step where a possible candidate topology or a population of topologies is created. In this research, neural network topologies are generated by defining graph grammar or graph transformation rules. These rules have the ability to generate all possible neural networks via a recognize-choose-apply cycle (RCA). The worth of each of the candidate topologies is then evaluated by measuring their ability to properly train to a given set of data. A genetic algorithm is utilized to train this data. While genetic algorithms provide a quick and easy way of training these topologies, they are not used to create the topologies. After all the candidate topologies are evaluated, further changes to the topologies are defined by the ***guidance method***.

A novel tree traversal method for searching for topologies is presented as our guidance strategy which efficiently considers only the feasible set of neural networks. The diagram shown in

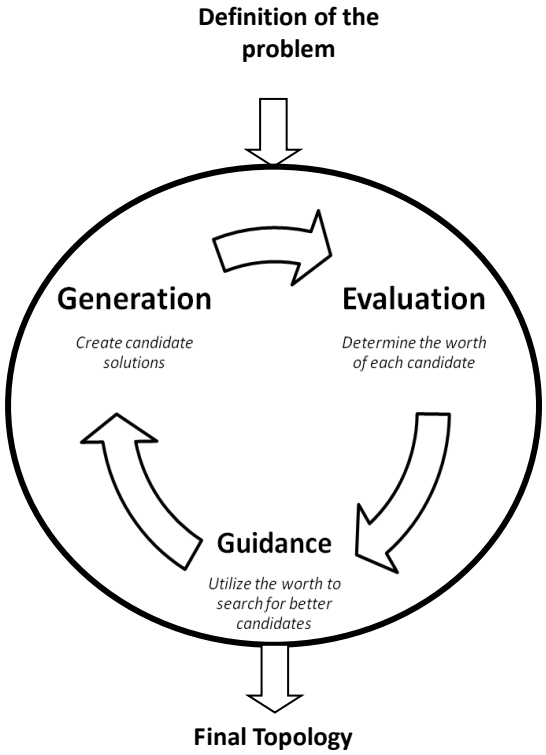


Figure 2: The method of neural network topology design

Figure 2 summarizes the method and the following sections describe each step in more detail.

3.1 Generation

Even though the theory behind graph grammars has existed for 40 years now, their utility in the context of design has been recognized only recently [15]. In the case of neural networks, graph grammars form a basis to verify architectures in a diagrammatic notation, and are viewed as a model to simulate dynamic evolution [16]. Graph grammars or graph transformation systems deal with ways to understand and define operations such as addition or subtraction of graphs. Applying a graph grammar to a specific problem involves representing the problem in the form of graph. Design of any candidate solution to the problem originates from a seed. The seed is usually the most basic graph and contains user defined information about the problem. In the case of neural networks this information is the number of input parameters and the number of responses. Hence, the seed chosen for generation of neural networks is a graph consisting of input nodes and output nodes with no connections between them. An example seed for a problem with one input and one output is shown in Figure 3.

A graph grammar consists of a set of rules. The rules are formulated prior to the design process to capture basic design knowledge of that particular design problem. A neural network generally has of a set of interconnected nodes organized in layers between the input and output nodes. This forms the underlying knowledge for generating neural networks that is to be captured by the rules. A rule has a left hand side which defines the conditions in which the rule is applicable and right hand side which describes how the graph is transformed when the rule is applied. To generate neural network topologies, six rules shown in Annex A, are defined to capture the transformation from the seed to valid topologies. The figure illustrates a list of rules and an example of how the rule transforms an arbitrary graph. Rule 1 and rule 2 initiate the process of generating a topology. These rules ensure that there are no unconnected nodes. Rule 1 connects two unconnected nodes while rule 2 adds one node in the hidden layer. Rule 3, a variation of rule 2, adds a node between connected nodes. Rule 3 creates a new hidden layer. Rules 4 and 5 increase the interconnectivity between the nodes. Rule 6, which also adds a node, differs from rule 3 in that it adds a node in parallel thus increasing the number of nodes in a particular hidden layer.

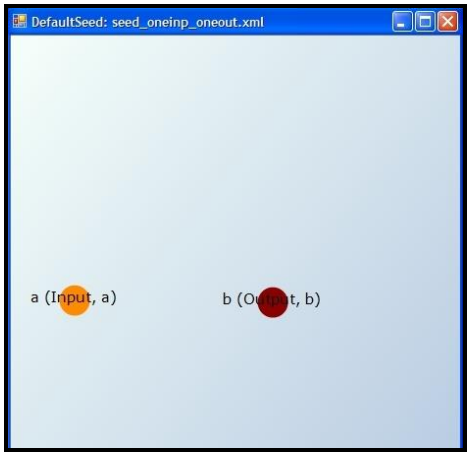


Figure 3: A seed with one input and one output

A complete graph or a candidate topology is generated by the application of these grammar rules iteratively via a three step process: recognize, choose and apply [17]. By applying each rule consecutively a candidate usually becomes more complex and comes closer to being a complete solution. At the start of each iteration, *recognize* is the step where all possible locations in the graph are identified where the set of rules can be applied. These locations define a list of all possible transformations that can be applied on that graph at that stage. Given the list of rule choices, the second *choose* step – is the stage where the design agent (be it human or computer) selects a particular rule that would make the best modification to the graph. This rule is now *applied* on the graph and the graph is transformed to a new state. Figure 4 summarizes the three step process with an example. Three rules are recognized at different locations in the graph as shown. Rule 4 is chosen and the effect of the application of the rule – an added hidden node in the second layer is also shown.

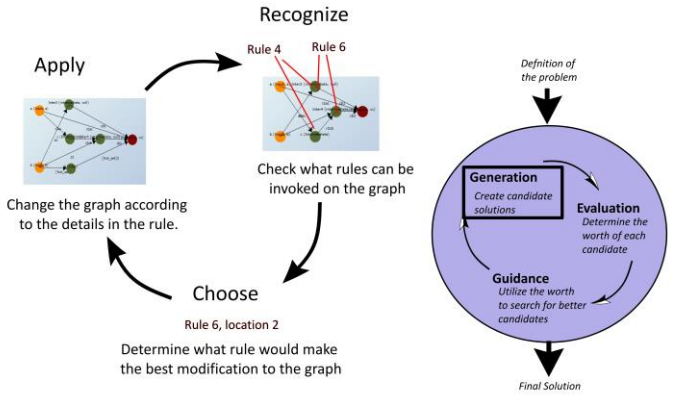


Figure 4: Generation: the recognize choose and apply cycle

The final candidate obtained depends on the choice of rules applied. It is the application of rules that transform seed into the solution. The design agent must be knowledgeable enough to predict how the design will be affected and judge the benefit in invoking one rule over another. If the lack of concrete knowledge prohibits the design agent from making an intelligent choice then finding the solution is simply a matter of exhaustively searching through a tree of candidate solutions that result from repeated application of rules on the seed based on the performance of the candidate. This is described in the following sections.

3.2 Evaluation

The performance of any neural network is specified by its ability to train or estimate the given data. It should also properly approximate or predict responses for unforeseen data from the same data set. The ability to do this is referred to as generalization of the estimation. The ability to train is measured by the training error and the generalization is measured by generalization error. Training error is measured by the root mean square of difference between the response predicted by the network and the actual value. It has been commonly found that as the complexity of the topology increases the training error decreases. The generalization error decreases with an increase in the complexity until an optimum topology is obtained, after which further increases in complexity of topology will cause the generalization error to increase due to overfitting.

To train the neural network topologies created in the generate step, we utilize a genetic algorithm. The motivations behind the selection of a genetic algorithm are its ability to avoid a local minimum and its reported success in training the neural networks [8]. These two benefits are extremely useful when dealing with networks that are partially connected as they usually take a lot of time to train properly using traditional methods such as backpropogation. The error obtained from a genetic algorithm is the basis for further topological changes. Once a suitable topology, one which satisfies user acceptable error limit, is found other methods of training – backpropogation or a hybrid of backpropogation and genetic algorithm could be used to train the neural network more accurately.

3.3 Guidance

The problem of finding an optimum topology is now formulated as a tree search. The tree results from application of all possible rules on the seed as shown in Figure 5. It is to be noted that the purpose of guidance is how to effectively search this tree to find the optimum topology; it is not necessary to generate all the solutions of this tree.

Each state on the tree represents a feasible candidate topology. It is intuitive to imagine a tree traversal search that is guided by the training error and the generalization error. The objective of such a search would be to find the candidate with the lowest training and generalization error. Another approach would be to make the design of topologies an interactive process with the user giving feedback about the topology and guiding further generation. Both these methods are discussed below.

A unique population based fully automated tree traversal search process has been implemented for searching through the tree. The search starts with an initial population of candidate topologies. This initial population is feasible – generated by applying rules until there is a connection between all the input and the output nodes. A candidate topology generated automatically by applying rules at random on the seed. Each candidate topology in the population is trained using a genetic algorithm. A topology's survival in the population depends on its training and generalization error. A portion of topologies which showed the largest error are removed from the population. New topologies are added to the remaining population by applying a new rule, selected at random, on the remaining good topologies. Some topologies are also generated

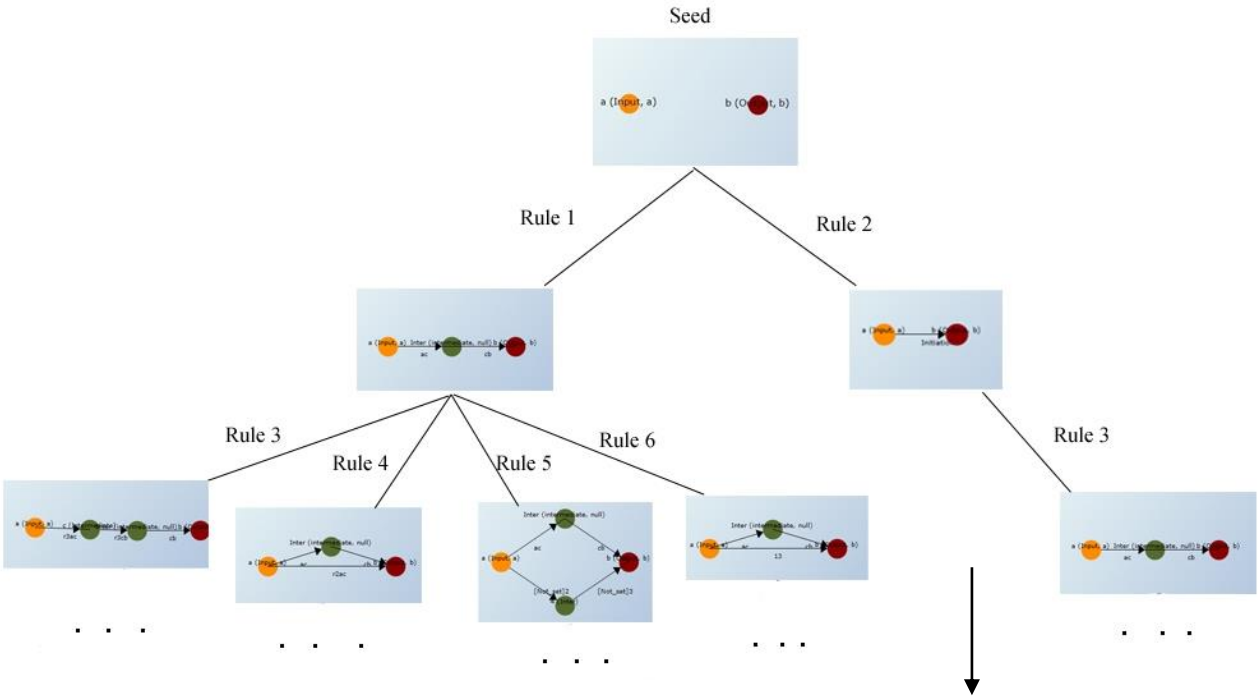


Figure 5: Tree resulting from application of different rules on the seed

by removing or undoing an applied rule. The rest of the population is filled up by new topologies. This process is implemented iteratively and the average training and generalization error of population is recorded. The search is stopped when the average error of the population increased continuously for five successive iterations. The best topology is one with the least error in the population.

In addition to the fully automated approach, a second approach is to make the search interactive. In this interactive method, the user is presented with set of rules that can be applied to the graph since they have been recognized in the ***recognize*** step of generation (Figure 2). The user analyzes the result of each rule and makes the choice as to which rule to apply. After each rule application the topology is trained quickly and automatically and the user is presented with the training and generalization error of the topology. Based on this information, the user may choose to undo the last applied rule or apply a new rule. Essentially, at any stage the user may undo any number of rules applied and continue from there. A good topology is reached on when a user is satisfied with the error obtained.

4 RESULTS

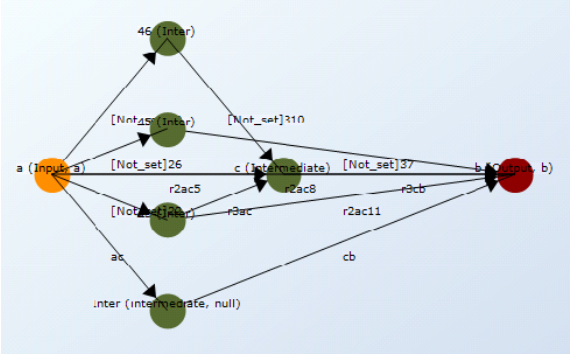
Our approach is implemented on a standard desktop computer with an AMD Athlon Dual-core 2.21 GHz processor with 1 GB RAM. To design topologies using a graph grammar we make of use GraphSynth [18]. GraphSynth is the first publicly available approach to creating graph grammars. It is an open-source, intuitive and extremely flexible platform for implementing graph transformation systems. Using GraphSynth, one can design, implement, test, and automatically invoke grammar rules. Our approach was tested on the following problems. All the topologies use a sigmoid activation function within each node.

4.1 Sine function with interactive user assisted search

A set of 200 data points was generated using the test function (originally used in [19]):

$$y=\sin(\pi x) + \text{noise}, \quad x \in [0,1] \qquad (2)$$

A subset of 170 points was used for training and the remaining 30 points were used to evaluate the generalization error. The search process in the user-assisted generation of neural network relies on the user to make the choices of rules to be applied to generate the neural network topology. Several trials with different users were conducted. Satisfactory topologies were obtained in all the trials. The number of rules applied and the number of times the users choose to undo the rules applied differed from user to user. Annex B shows the choices made by two users and the errors at each step. Figure 6 shows the final topology obtained by user 1 and the plot of its response compared to the actual value. It can be inferred from the table that the user's decision to undo or apply rules clearly depended on the need to decrease training error and generalization error. These sequential choices appear those of like a steepest descent algorithm where the user makes perturbations around the space and chooses one with the least error. The users were also influenced by their previous choices that had yielded good results. Furthermore, the users complained about the fact that it took a long time to know the result of a choice made since after each choice the resulting topologies had to be trained before the errors were presented. While the user assisted search produced some good results, the feedback seems to emphasize the need for an automated approach for searching topologies. For problems with more inputs and outputs, the human interactive approach is not practical.



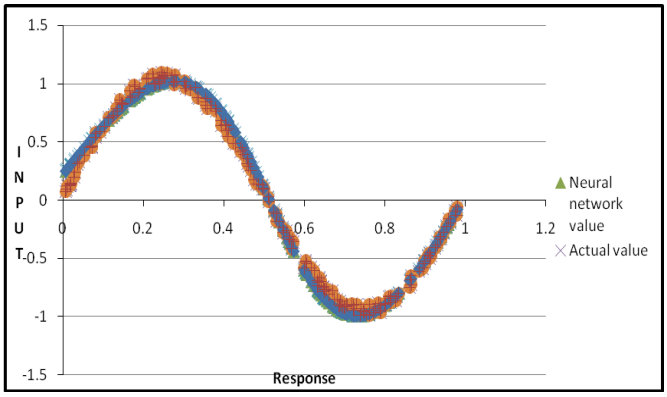


Figure 6: Final topology obtained by user #1 and plot of its response with actual value.

Trial	*Training error*	*Generalization error*
1	0.016	0.349
2	0.011	0.223
3	0.020	0.557
4	0.017	0.601
5	0.015	0.358
Average	0.015	0.417
STD dev.	0.003	0.157

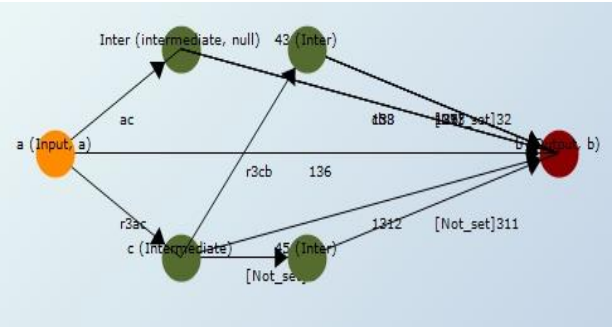
Table 1: Training and generalization error of various trials

4.2 Sine function with fully automated method

In this test an automated search was used to search for a topology. The data used in this test was the same used in 4.1. The search process was conducted with a population of 10 topologies. Each topology was trained for 100 seconds using a genetic algorithm. The optimal topology obtained with this approach differed in each of the 5 trials conducted even though very good candidate were found in almost all cases. Table 1 shows the training and the generalization errors of each of the trials. Figure 7 shows the final topology obtained in trial 1 and compares the output of the candidate with an exact plot.

To illustrate how topology of neural network affects its performance, a topology that has poor response is shown in Figure 8. This topology had a training error of 0.047 and a generalization error of 1.200. It is to be noted that this topology was not obtained as the best solution in any of the trials, but is chosen randomly here to illustrate the point. It can be observed that the topology does not have enough nodes to accurately map the function. This topology underfits the data due to its few hidden nodes and resulting insufficient complexity.

Figure 9 shows the convergence of the search process on the topology. The errors represent the average error of the population and not the best solution. Note how efficently the search process functions. With only 10 candidates per population, a sucessful solution is found in fewer than 10 iterations. As the number of iterations increase the average training error of the population decreases until iteration 7 after which it increases. The generalization error also follows the same trend. After iteration 7 further topological changes tend to overfit the data, hence there is a increase in the error. These graphs also bring to light the obvious relationship between a topology and performance.



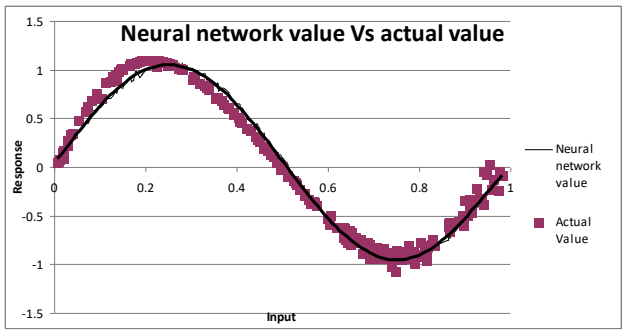
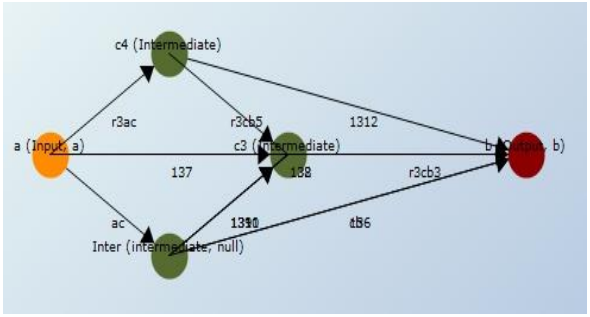


Figure 7: The final topology obtained and plot comparing its response with exact values



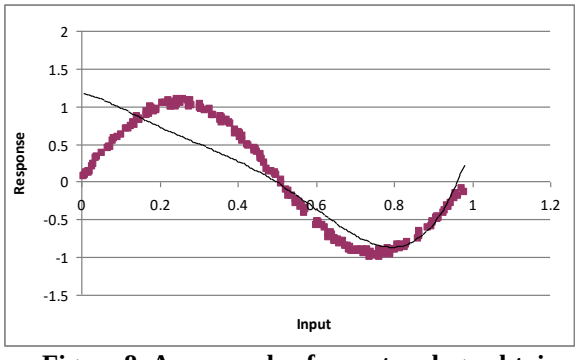
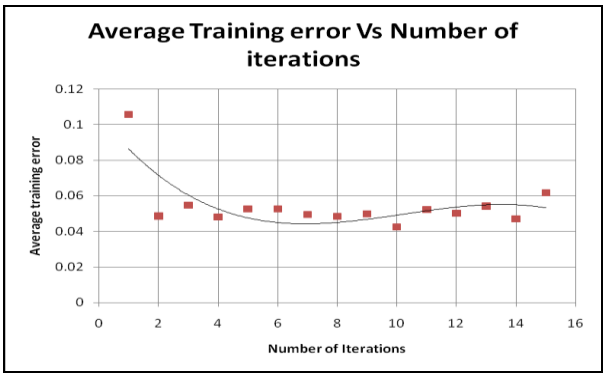


Figure 8: An example of poor topology obtained and a plot of its response



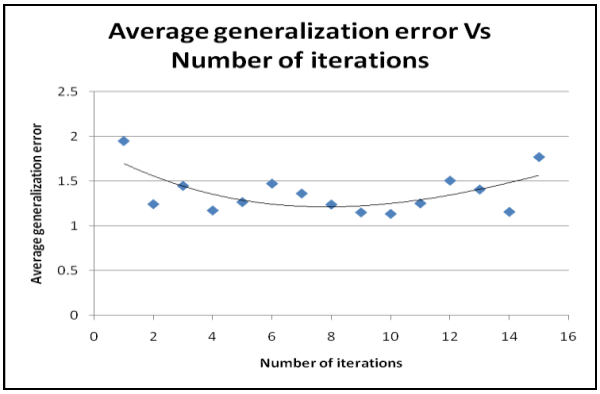


Figure 9: A plot of the training error (top) and generalization error (bottom) vs. the number of iterations of the search process

4.3. Rosenbrock's banana function

In this experiment the algorithm tries to find the optimum topology for a data generated by the function

$$y=100.\ (x_2-x_1^2)^2+\ (1-x_1)^2\ +noise;\ x_1,x_2 \in[-1,1] \qquad (3)$$

The data consisted 250 points with 220 points used for training and 30 points are used for calculating the generalization error. The automated search approach was used with a population of 10 candidates and a training time of 100 seconds. The final topology obtained for this experiment is shown in Figure 10. The final topology has a training error of 0.09 and a generalization error of 0.11. Notice that the topology generated is more complex than that generated for the sine function. The surface plots comparing the response of the topology with the actual plot are shown in figure 11. Neural network almost exactly replicates the contours of the function.

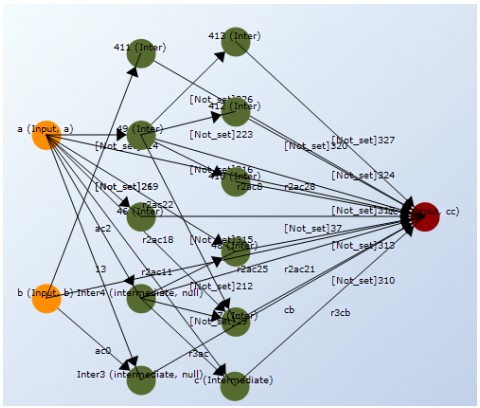


Figure 10: The final topology obtained

5 CONCLUSIONS

A simple, efficient and robust method of neural network topology generation is presented. The approach has the ability to generate both simple and complicated neural networks. Since the algorithm begins with a simple network and builds up to an optimum topology by applying rules, it efficiently generates a pruned network. While it appears that the current method has the ability to generate all possible neural network topologies,

Plot of the response of final topology *Actual plot of the function*

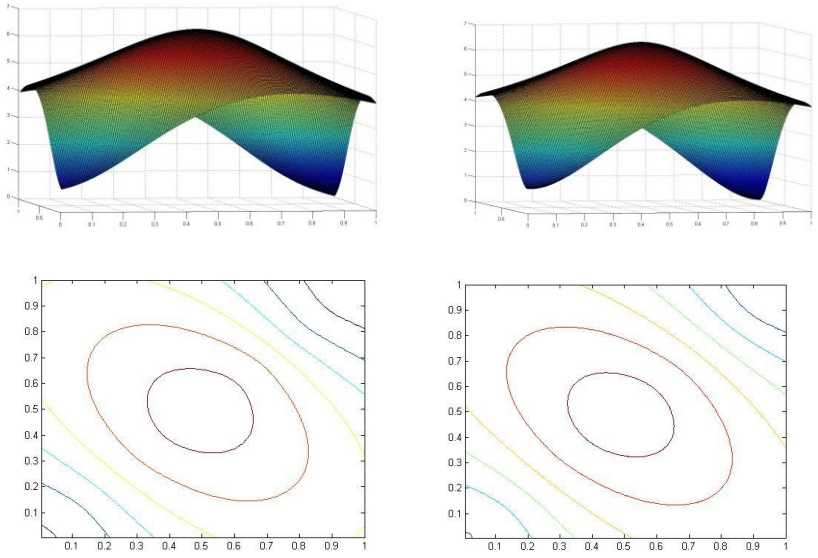


Figure 11: A plot comparing the response of final topology with the actual function

further research needs to be conducted to prove this claim. Further, this methodology of generation of neural networks always ensures a feasible topology. Feasibility is guaranteed by the fact that only valid topological transformations are captured by the rules at each stage. Rule applications at locations which result in an infeasible topology are not recognized. Hence the method is perfectly suited to evolving neural network topologies.

Previous implementations of generating neural network topologies make simplifying assumptions that result in a failure to explore the complete space of possible neural networks. These methods also need to deal with generating, evaluation and managing the infeasible topologies. Further, our approach of generating neural network topologies is faster than implementations using a GA. The elegant search method presented in this paper not only limits the search to the feasible domain but also can function effectively with a very small population. Approaches using GAs on the other hand use large populations and substantial number of iterations. Our results indicate that there is a definite mapping between topology and performance. Current research is being undertaken to develop a more efficient search process that utilizes this information. Future research is aimed at developing a more advanced guidance strategy to search though the space. This search space increases exponentially as the number of inputs and outputs increase. Based on the results obtained using the interactive user search approach, it is easy to imagine an approach which uses a greedy search as the guidance strategy. This would involve choosing the rule which would reduce the error the most. The current research looks very promising and should extend well to solve more complicated problems.

REFERENCES

[1] Slawomir W. Stepniewski and Andy J. Keane, "Pruning backpropagation neural networks using modern stochastic optimization techniques". Neural computing and applications, vol. 5, no2, pp. 76-98 1997.

[2] Sarle, W.S., ed. (1997), Neural Network FAQ, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: ftp://ftp.sas.com/pub/neural/FAQ.html

[3] Andrés Pérez-Uribe, Structure-Adaptable Digital Neural Networks, Thesis No 2052, École Polytechnique Fédérale De Lausanne.

[4] Xin Yao, "Evolving Artificial Neural Networks", Proceedings Of The Ieee, Vol. 87, No. 9, September 1999

[5] Fahlman S. E., Lebiere C., "The cascade-correlation learning architecture", Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[6] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize", Neural Networks, vol. 4, no. 1, pp. 67-69, 1991.

[7] Russell Reed, "Pruning Algorithms-A Survey", IEEE Transactions On Neural Networks, Vol. 4, No. 5, Seyember 1993.

[8] Richard K Belew, John McInerney and Nicol N Schraudolph, "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", CSE Technical report # CS 90-174, University of California at San Diego.

[9] Lucas, S., 1995. Growing adaptive neural networks with graph grammars. In Proc. of European Symposium on Artificial Neural Networks, pages 235-240.

[10] Antonia J. Jones, Genetic algorithms and their applications to the design of neural networks, Neural Computing & Applications, 1(1):32-45, 1993.

[11] Fiszelew, A., Britos, P, Perichisky, G. & García-Martínez, R. "Automatic Generation of Neural Networks based on Genetic Algorithms", Intelligent Systems Laboratory. School of Engineering. University of Buenos Aires. Paseo Coln.

[12] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," Neural Computation, vol. 5, pp. 54--65, 1994.

[13] Kenneth O. Stanley and Risto Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", Evolutionary Computation 10(2): 99-127, MIT Press 2002.

[14] Timothy Andersen, Kenneth O. Stanley, and Risto Miikkulainen, "Neuro-Evolution Through Augmenting Topologies Applied ToEvolving Neural Networks To Play Othello", Independent study project and Honors thesis project, Department of Computer Sciences, The University of Texas at Austin.

[15] E. K. Antonsson and J. Cagan, "Formal Engineering Design Synthesis", Cambridge University Press, 2001.

[16] Jun Kong, Kang Zhang, Jing Dong, Guanglei Song, "A Graph Grammar Approach to Software Architecture Verification and Transformation", Computer Software and Applications Conference, 2003.

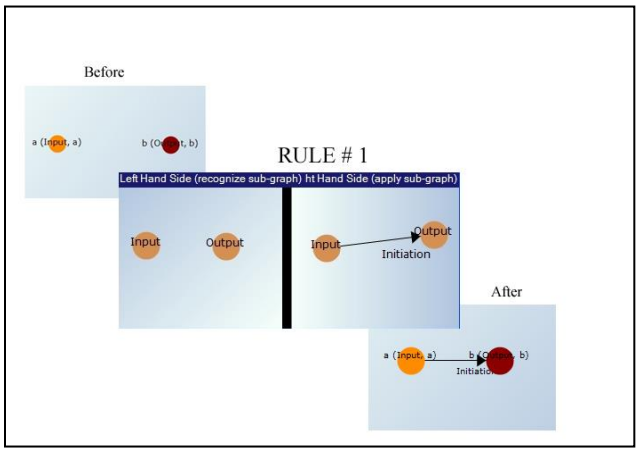
[17] Campbell, M.I. "A Graph Grammar Methodology for Creative Systems," Artificial Intelligence, in review 2007.

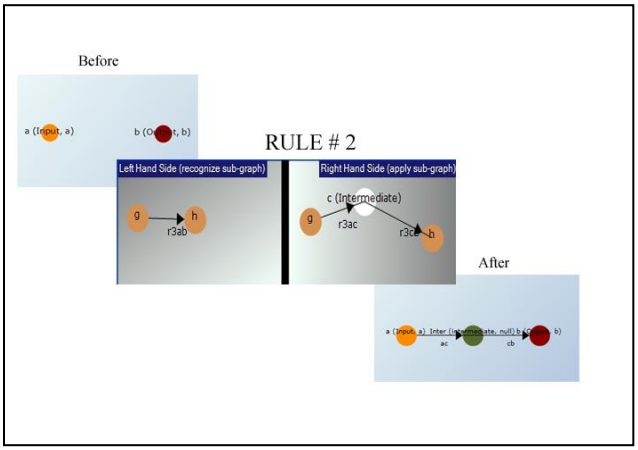
[18] M. I. Campbell, "The Official GraphSynth Site", http://www.graphsynth.com, University of Texas at Austin, 2006.

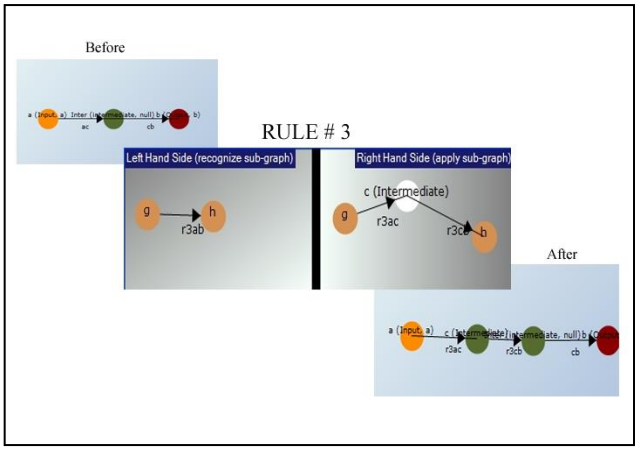
[19] Matteo Matteucci, "ELeaRNT: Evolutionary Learning of Rich Neural Network Topologies", Center for Automated Learning and Discovery, Carnegie Mellon University, Technical Report N. CMU-CALD-02-103.

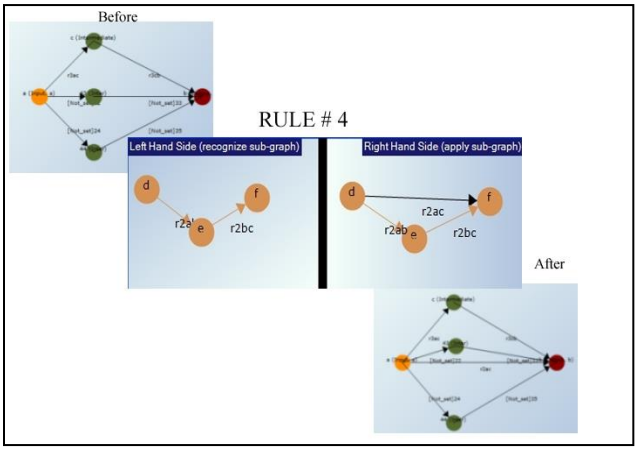
ANNEX A

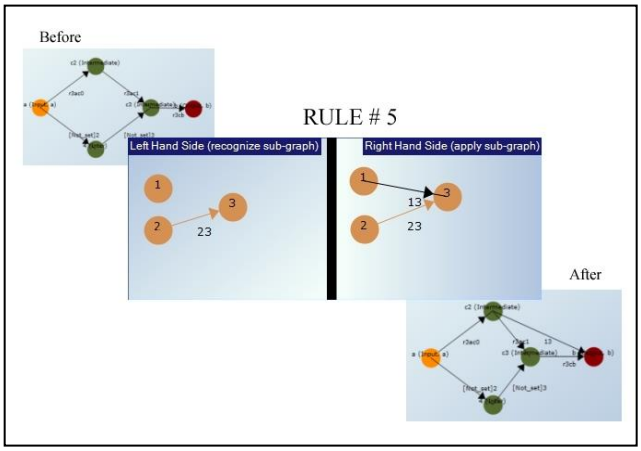
THE RULES DEFINED TO GENERATE NEURAL NETWORKS

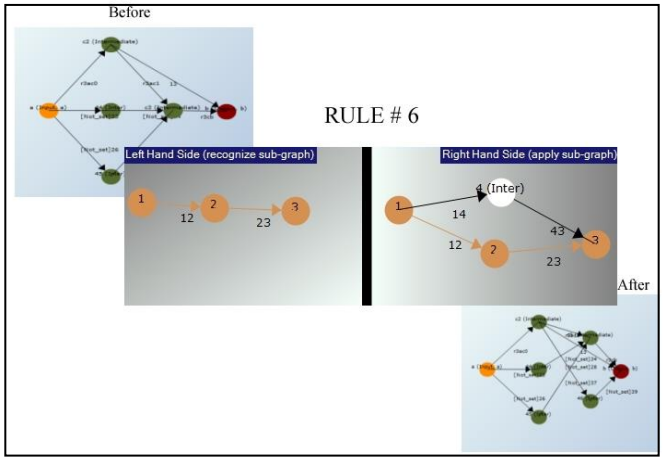












ANNEX B

USER CHOICES AND ERROR OF EACH CHOICE IN USER ASSISTED SEARCH

Iteration	*Rule applied*	*Training error*	*Generalization error*	*Rule applied*	*Training error*	*Generalization error*
	USER #1			USER # 2		
1	Rule 1	0.077	1.423	Rule 1	0.0773	1.422
2	Rule 4	0.067	1.350	Rule 5	0.073	1.360
3	Rule 5	0.073	1.355	Rule 4	0.069	1.278
4	Undo Rule 5			Undo rule 4		
5	Rule 3	0.067	1.230	Rule 3	0.077	1.428
6	Rule 4	0.064	1.192	Undo rule3		
7	Rule 6	0.043	0.789	Rule 3	0.072	1.349
8	Undo Rule 6			Rule 6	0.070	1.288
9	Rule 6 at another location	0.031	0.559	Rule 6	0.063	1.157
10	Rule 3	0.044	0.799	Undo Rule 6		
11	Undo Rule 3			Rule 3	0.071	1.326
12	Rule 6	0.018	0.281	Rule 6	0.064	1.184
13	Rule 5	0.010	0.210	Rule 3	0.064	1.181
14	Search ended by user			Rule 6	0.038	0.710
15				Undo Rule 6		
16				Rule 6	0.051	0.946
17				Undo Rule 6		
18				Rule 6	0.035	0.647
19				Rule 6	0.029	0.596