NVIS: an interactive visualization tool for neural networks

Matthew Streeter[a], Matthew Ward[a], and Sergio A. Alvarez[b]

[a]Worcester Polytechnic Institute, Worcester, MA 01609
[b]Wellesley College, Wellesley, MA 02481

ABSTRACT

This paper presents NVIS, an interactive graphical tool used to examine the weights, topology, and activations of a single artificial neural network (ANN), as well as the genealogical relationships between members of a population of ANNs as they evolve under an evolutionary algorithm. NVIS is unique in its depiction of nodal activation values, its usage of family tree diagrams to indicate the origin of individual networks, and the degree of interactivity it allows the user while the learning process takes place. The authors have made use of these features to obtain insights into both the workings of single neural networks and the evolutionary process, based upon which we consider NVIS to be an effective visualization tool of value to designers, users, and students of ANNs.

Keywords: neural networks, visualization, graphs, evolutionary computation

1. INTRODUCTION

1.1. Introduction to Artificial Neural Networks

Artificial neural networks are models of adaptive distributed computation with roots in neurobiology. ANNs have been used with great success in a variety of domains to solve problems involving function approximation or classification[1-3], including problems of face recognition[4], engine control[5], and learning to drive a car on public highways[6]. In its simplest form, a feedforward ANN can be considered to be a weighted directed acyclic graph with specially labeled node sets for input and output signals. Application of specific signal values to the inputs of an ANN evokes a specific activation level at each node of the ANN; this activation level is determined by the input values to the network and by the activation levels of the nodes feeding into the given node, weighted by the weights of the connections between nodes. In this way, the nodes of an ANN operate collectively to define a nonlinear mapping from inputs to outputs parameterized by the connection weights.

Adaptation in ANNs consists of a suitable strategy for adjusting the weights in a way that optimizes some domain-dependent measure of performance. Strategies for supervised adaptation in ANNs include error backpropagation[7-8] and evolutionary computation[9-12], both of which are implemented in this system. In the latter approach, weights from different individuals of an entire *population* of networks are combined in a process that resembles reproduction, complete with operators for recombination, mutation, and subsequent selection based on performance. This approach is described further in the next section.

1.2. Introduction to Evolutionary Computation

Evolutionary computation is a technique used to search for an instance of an entity that satisfies some desired criterion, where the design of the search process is intended to imitate natural evolution. Using an evolutionary algorithm requires one to define an appropriate representation or "chromosome", a function that evaluates the "fitness" of a particular chromosome, and operators that breed (in general) two chromosomes to form a child. In the case of evolving neural networks, the chromosome is the array of weights associated with the network, and the fitness function is typically taken as the root-mean-squared difference between the network's expected and actual output over a given set of examples of the problem to be solved. In our system, two networks are bred by averaging their corresponding weights. The particular approach to evolutionary computation adopted in this system is the *evolution strategy* algorithm[9]. Briefly, this algorithm involves evolution over multiple generations of a fixed-size population producing a fixed number of children, where the members of each subsequent generation are selected from either the parents and offspring of the previous generation, or from the offspring alone. In all the experiments reported in this paper, selection is from the former set; the notation ($[\mu]+[\lambda]$) is used to indicate the use of such an algorithm with μ parents and λ offspring. The evolution strategy approach also makes use of a special coefficient associated with each gene in the chromosome (in this case, each weight) which determines the degree of mutation that this gene undergoes during a breeding operation, and which we later refer to as the *volatility* of a particular weight.

1.3. Relation to Previous Work in Neural Network Visualization

Existing techniques for visualization of neural networks include Hinton diagrams and Bond diagrams[13]. Hinton diagrams consist of a number of sets of boxes, with each box representing a single weight, organized so as to allow one to infer the network topology. In Hinton diagrams, the area of each box is proportional to the absolute value of the corresponding weight, while the color of the box is used to indicate the sign of the weight: white for positive, black for negative. Though the relative magnitudes of different weights are readily apparent from these diagrams, Hinton diagrams fail to give a direct (rather than inferred) depiction of network topology. Bond diagrams improve on Hinton diagrams by superimposing a visual depiction of connection weights on a planar projection of the network graph, using triangles of various sizes to represent the weights, with the altitude of each triangle being proportional to the absolute value of the corresponding weight. However, neither of these approaches includes a mechanism for the visualization of nodal activation values, which indicate how a neural network processes a particular set of inputs. As will be seen in section 2, "Feedforward neural network visualization", NVIS provides a neural network visualization which attempts to overcome both of these limitations, and in addition provides a representation of the volatility parameter associated with each weight, as used by the evolution strategy algorithm. Furthermore, NVIS addresses the visualization of genealogical relationships arising from the evolutionary adaptation process, both in the form of the "Generations" window described in section 4 and the "Family Tree" window described in section 5. Finally, NVIS allows the user to interactively adjust training parameters during adaptation and makes the results of this interaction immediately available to the user in visual form, thus providing important cues that aid in understanding the process and results of adaptation.

2. FEEDFORWARD NEURAL NETWORK VISUALIZATION

The primary and most important visualization developed as part of this tool is that used to depict a single neural network, as illustrated in Figure 1 (color versions of this and all other figures in this paper are available online at http://davis.wpi.edu/~matt/projects/NVIS/colorfigures.html). The intent of this visualization is to quickly and intuitively convey the weights, volatilities, and activation values associated with a particular network. Figure 1 depicts a feedforward neural network drawn from a population trained to determine the parity of a four bit number, presented both as a weighted graph and in a compact matrix form. This section describes the weighted graph representation; the compact matrix representation will be dealt with in section 3.

The weighted graph representation consists of a number of black circles, each of which represents a node, a set of line segments that connect nodes, representing weights, and a short bar running perpendicular to each weight, representing its volatility. Each row of circles represents a layer of nodes in the network, with the top row representing the input layer, the bottom row the output layer, and all other rows corresponding to hidden layers. The network depicted in Figure 1 has four inputs, one output, and two hidden layers of two nodes each.

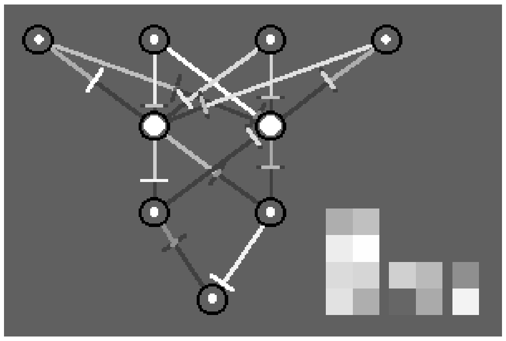
Figure 1. Network trained to determine the parity of a four bit number.

In this representation, the diameter of the white circles inside each node has been mapped on a sigmoid scale to the node's activation value. Though it cannot be seen in the grayscale illustration of Figure 1, the values of the weights have been linearly mapped to both the length of the colored portion of the line segment and the brightness of the color in which it is drawn, with the hue of this color encoding the sign of the weight: cyan for positive and red for negative. A similar length and coloring scheme has been used in the short, perpendicular bars to indicate weight volatility, only without the mapping to hue, since volatility is an unsigned quantity. Note that weights are a fixed property of an individual network, whereas activation values are a function of the particular inputs for which the network is being run. In general, there will be many training examples (and therefore many different sets of inputs) available for use in evaluating a network. In our system, the user may use a horizontal scroll bar to browse through the set of available training examples, or may select a specific training example by typing in the appropriate index.

The weighted graph representation in Figure 1 is part of a "Network Editor" window accessible for any network in the population history. This window allows the user to load and save networks, to load new sets of problem examples on which to evaluate a network, to refine a network via error backpropagation, and to manually adjust the weights using each colored bar as a slider. When running error backpropagation, the changes made to the weights by the algorithm are tied to the visualization, thus providing an animation of the training process for this particular algorithm.

3. COMPACT MATRIX REPRESENTATION

The three colored matrices depicted in the lower right corner of Figure 1 and reproduced in Figure 2 below represent an enlarged version of the what we refer to as the "compact matrix representation" for the network discussed in the previous section. The purpose of this representation, which can in principle be displayed using only a single pixel for each colored square, is to allow many networks to be displayed on the screen at once. The correspondence between matrix squares and network weights (which are drawn in the same color in both representations), is designed to have a special property related to network execution, and is defined formally below.

Matrix n, starting from n=1 on the left, represents the set of weights running from layer n to layer n+1 in the network, with n=1 corresponding to the top layer. The color used to fill the square in row y, column x of matrix n represents the weight of the connection from node y of layer n to node x of layer n+1, and is the same color that is used to draw the colored portion of the line segment representing this weight. This representation has the convenient property that the multiplication of the matrices in the compact representation yields the final transformation matrix for a network using the linear activation function F(x)=x.

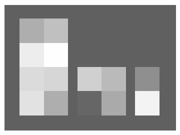
Figure 2: Compact matrix representation

We note that this compact matrix representation bears some resemblance to the multidimensional data set visualization technique employed by Beddow[14], although in our case the representation is more directly related to the known structure of the networks being considered.

4. DEPICTION OF POPULATION HISTORY

Using the compact matrix representation allows many networks to be displayed on the screen at once. In Figures 3 and 4, we present the output of two visualizations designed to take advantage of this: the "Generations" window (Figure 3), which organizes all the networks in the population history by generation, and the "Family Tree" window (Figure 4), which depicts the family tree for a specific network. The "Generations" window is discussed in this section; the "Family Tree" window will be discussed in the next.

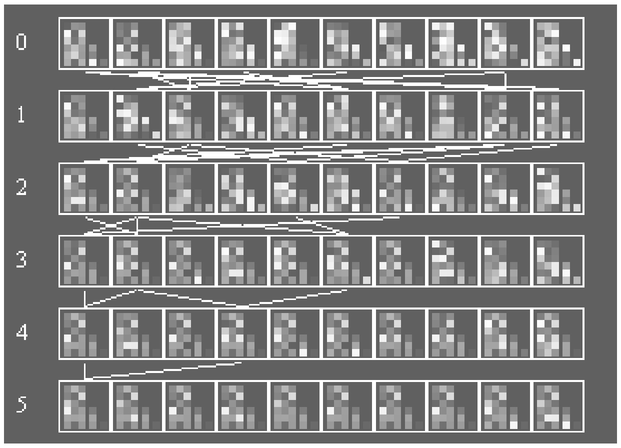

Figure 3. First six generations of a pool of networks trained to predict CPU performance.

Figure 3 illustrates the first 6 generations of a pool of networks trained to predict the performance of a CPU[15] based on six inputs, including cache size, main memory size, and clock rate. Each generation is represented by a row of compact matrices, presented from left to right in decreasing order of fitness. The user may select any of the networks in the population history by clicking on the corresponding matrix. Once a network has been selected, menu options are available to bring up the "Network Editor"' or "Family Tree" windows for the selected network. Additionally, when a network is selected, a series of links are drawn within the "Generations" window to connect the network to its ancestors. Though it cannot be seen in the grayscale illustration of Figure 3, green lines are used in this visualization denote parentage, while white lines indicate survival from one generation into the next.

5. HEREDITY VISUALIZATION

Figure 4 illustrates the family tree for the network selected in figure 3. The organization of this diagram follows that of a standard genealogical tree, with the selected network at the root (lowest position), its two parents connected to it by lines above, and each parent displayed as the root of its own family tree, recursively. Each row of compact matrices is presented at the maximum allowable size (pixels per colored square) given the available window width and the number of networks that must be displayed. Though this example is well-behaved in the sense that all ancestral lines have the same length, and therefore reach the same height in the diagram, it is possible using a $(\mu+\lambda)$ evolution strategy for two parents of the same network to originate in different generations, so that in general the height of one parent's family tree will not be equal to that of the other. In such cases, NVIS simply allows the branches of the tree to reach variable heights, so that equal height corresponds to an equal ancestral depth, rather than an equal age.

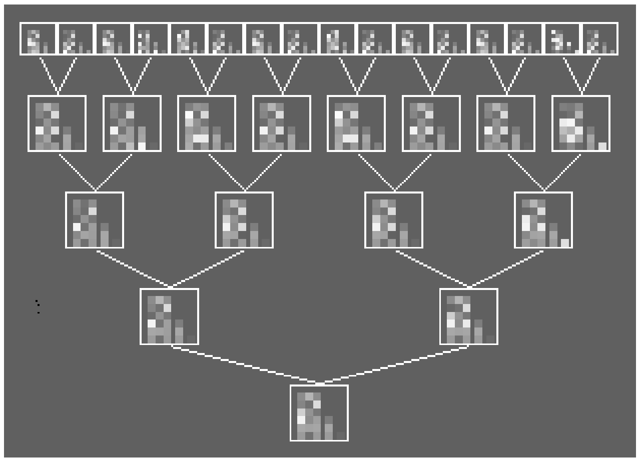
Figure 4. Family tree for selected (lower left) network in Figure 3.

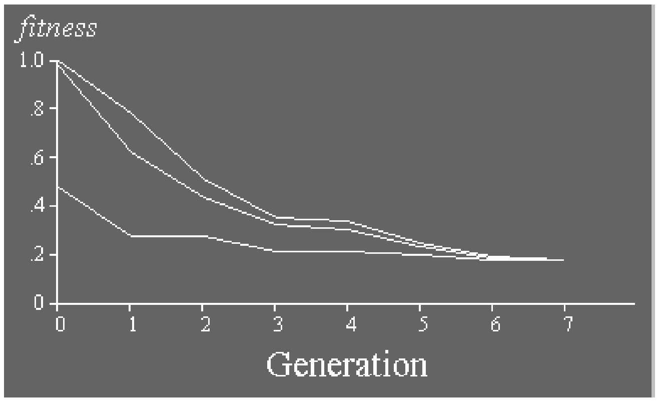

Figure 5: Fitness graph for first 8 generations of population of networks trained to approximate a sine wave.

6. AN INTERACTIVE ENVIRONMENT

The importance of interactive visualization in machine learning applications has been recognized by Koza[16], the inventor of genetic programming, and by Robertson[17]. The environment presented to the user by the NVIS system is designed to be highly interactive, with a large amount of information available to the user at any moment during the course of the evolutionary process. In addition to the "Network Editor", "Generations", and "Family Tree" windows described above, NVIS provides a simple 2D plot (see Figure 5 for grayscale illustration) of the best, worst, and average fitness of each generation as a function of time, graphed in green, red, and blue, respectively, which allows the user to easily gauge the progress of the evolutionary algorithm. Thus, while the "Network Editor" and "Family Tree" windows described above provide the ability to examine in detail an individual network or hereditary line of networks, the "Generations" and fitness graph windows serve as effective summaries of the evolutionary process as a whole. Furthermore, the ability to load and save networks through the "Network Editor" window encourages comparison of trained networks across different selections of evolutionary and topological parameters, over different sets of problem instances, and over multiple evolutionary runs against the same problem. These features combine to create an environment that actively encourages the user to monitor and participate in the learning process, rather than treating the evolutionary algorithm and the trained networks it produces as a black box.

As in a standard GUI program, each of these windows can be dismissed, invoked, resized, and repositioned as desired. For ease of use, all windows and parameter settings are accessible through a central control panel, a screen shot of the which is reproduced in Figure 6. Note that in this control panel, each bold word is also a "fly-over" button; when the screen capture for Figure 6 was performed, the mouse pointer was hovering over the "Pause" button.

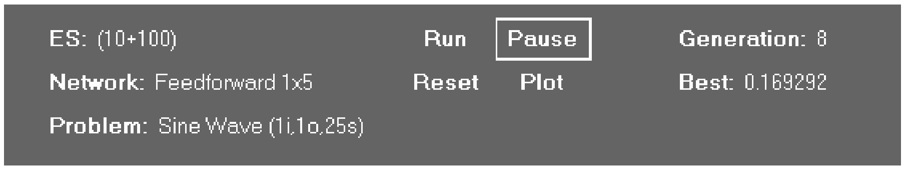
ES: (10+100)
Network: Feedforward 1x5
Problem: Sine Wave (1i,1o,25s)
Generation: 8
Best: 0.169292

Figure 6: The NVIS control panel.

Below we present a set of steps constituting a typical "use case" for the NVIS system. These steps are intended to illustrate how NVIS could be useful for educational or tutorial purposes. More specific uses are described in section 7, "Results".

1. Load the file "sin.ts" from local directory. NVIS indicates that this training set consists of 25 examples, each with 1 input and 1 output.
2. Select an evolution strategy of (10+100) and a network architecture consisting of 1 hidden layer with 5 nodes, each of which use the sigmoid activation function, then begin the run.
3. After a few generations, bring up the "Generations" window. Examine the population history.
4. Select the best network in the latest generation, and bring it up in the "Network Editor" window. Examine its activation values for various training examples.
5. Attempt to refine this network using error backpropagation. Noting a moderate improvement in performance, save this network for possible later use.
6. Examine the real-time fitness graph. Observing that the evolutionary algorithm has plateaud, most likely indicating that the population has become homogenous. Verify this with "Generations" window.
7. Attempting to escape this local optimum, increase mutation rate with slider. Observe sudden dip in fitness graph.
8. Returning to "Generations" window, observe how one mutated individual changed the makeup of the entire population.
9. Using "Family Tree" window, trace origin of later generations to mutated individual observed in previous step.

7. RESULTS

After developing this system, the authors conducted a number of experiments solving various problems with neural networks. These included toy problems, such as approximating a sine wave, learning to square a number, and balancing a scale, problems of moderate difficulty, such as determining the parity of a 4-bit number, and difficult real-world problems such as predicting the average value of houses in a development, predicting the performance of a CPU, image recognition, and diagnosis of breast cancer. In solving these problems, we made use of the visualizations and interactive features of this tool, in some cases validating the approaches we had developed, and in some cases discovering unexpected uses of our system. Our experiments have found this visualization tool to be of use in four areas: manually designing networks to solve problems or refining networks that have already been trained, gaining insights into the shape of the error surface for a particular problem and network topology, understanding the phenomenon in evolutionary computation known as "genetic drift", and extracting problem-specific domain knowledge from the operation of a network. Each of these areas is elaborated upon below.

7.1. Manually designing and refining networks

The ability of each of the colored bars in the graphical network visualization to act as a slider allows the user to attempt to manually design a neural network from scratch or to refine an existing network trained through evolutionary computation or error backpropagation. Though manual design of neural networks to solve problems is for the most part impractical, such efforts may aid the understanding of the operation of a neural network in the case of a simple problem domain. For a network that has already been trained through evolutionary computation or error backpropagation, adjustment of individual weights and observation of the resulting change in fitness can give an idea of the relative importance of various weights in the network.

7.2. Insights into the shape of the error surface

A key factor in determining the success or failure of an evolutionary algorithm is the shape of the error surface in which the algorithm is supposed to find a global minimum. One simple insight into the shape of the error surface is provided by the fitness graph: namely, that if the fitness graph exhibits a plateau, it is likely that the error surface contains a large local optimum, from which the evolutionary algorithm finds it difficult to escape. In training neural networks to recognize winning board configurations in tic-tac-toe, for example, we observed the existence of several plateaus in the fitness graph, some lasting over 100 generations. Another interesting and unforeseen insight into the error surface is provided by sliding the weights of a network while running the backpropagation algorithm. The backpropagation algorithm performs a gradient-based search for a local optimum on the error surface. Therefore, when a weight is dragged to a new location while this algorithm is running, it will sometimes "snap back" to its original location as soon as it is released, as the algorithm returns to the same local optimum it was in before the weight was adjusted. At other times, however, the adjustment of one weight will cause many others to shift dramatically in response, as a new local optimum is found. It is thus possible using this tool to estimate the length of a local optimum on the error surface with respect to each of the axes in weight space.

7.3. Understanding genetic drift

Another use of this tool is in understanding genetic drift, defined as the tendency for all the members of an artificial population to converge to a single solution, in contrast to biological evolution, where many species compete within a single ecosystem. To understand why this phenomenon occurs, it is helpful to refer to Figure 7 (a reproduction of Figure 3), in which, after only six generations, all of the networks in the population can be seen to be very similar. In Figure 7, it can readily be seen that the initial population (generation 0), whose weights have been randomly initialized, contains a great deal of diversity. This diversity is maintained for the first four generations, after which a single network, though possessing only a minor fitness advantage over its neighbors, is rapidly able to dominate the population. The best (leftmost) network in generation 3 is clearly a parent of the best network in generation 4. Though it is not shown in the figure, this network is also a grandparent of all but the two worst networks in generation 5, and an ancestor of all the networks in all generations to come. Were more generations shown, we would see that the populations under this evolutionary algorithm tend to consist of a set of almost identical members, a few of which occasionally mutate into a slightly more optimal form, and rapidly convert the rest of the population into their own image.

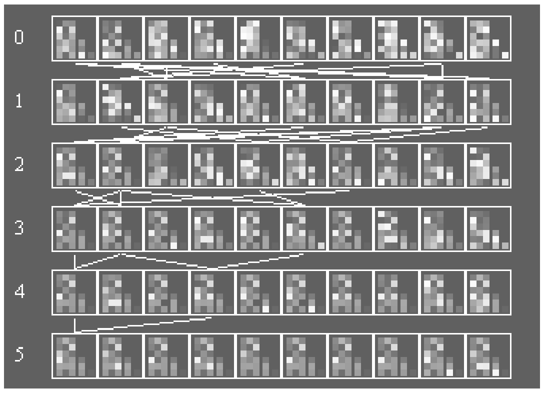

Figure 7. First six generations of a pool of networks trained to predict CPU performance.

7.4. Extracting problem-specific domain knowledge

The purpose of the weighted graph representation is to allow the user to understand how a network processes its inputs and produces its output. In doing so, the user may also gain insight into the problem the network has been trained to solve. An example of such a case is given in this section, where we discuss the results of analyzing a network trained to predict the average market value of houses in a development[15,18] based on 13 relevant inputs. The meanings of the 13 input attributes which define this problem are given in Figure 8.

Input	Meaning
1	Per capita crime rate (by town).
2	Proportion of residential land zoned for lots over 25,000 square feet.
3	Proportion of non-retail business acres per town.
4	Boolean variable equal to 1 if tract bounds the Charles River, 0 otherwise.
5	Nitric oxides concentration (parts per 10 million).
6	Average number of rooms per dwelling.
7	Proportion of owner-occupied units built prior to 1940.
8	Weighted distances to five Boston employment centers.
9	Index of accessibility to radial highways.
10	Full-value property-tax rate per $10,000.
11	Pupil-teacher ratio by town.
12	1000(P - 0.63)^2 where P is the proportion of blacks by town.
13	% lower status of the population.

Figure 8. Definition of inputs for housing value problem.

After running the evolution strategy algorithm for 6488 generations, we obtained the neural network depicted in Figure 9. Though it cannot be seen in the grayscale illustration of Figure 9, this network has the convenient feature that the weights emanating from the last two layers are all positive, so that the sign of the weights emanating from a specific input node gives a direct indication of the influence that node is having on the final output. Looking at input node (1), for example, which corresponds to the area crime rate, we saw that all the weights flowing out of the node are negative, which suggests that higher crime rates tend to reduce the value of a house. Likewise, examination of input node (6), the average number of rooms per dwelling, revealed that all the weights emanating from this input node are positive, suggesting that houses with more rooms tend to be worth more. In is also possible through examination of Figure 9 to gain an idea of the relative importance of each input by looking at the strength of the weights that emanate from the corresponding input node. This suggests, for example, that factors such as the proportion of homes built prior to 1940, the property tax rate, and the racial makeup of the community (inputs 7, 10, and 12, respectively) are not as important as the two zoning variables (inputs 3 and 4) or the student-teacher ratio (input 11) in determining the value of a house. In principle, this type of analysis could be applied to a problem domain for which no a priori knowledge exists.

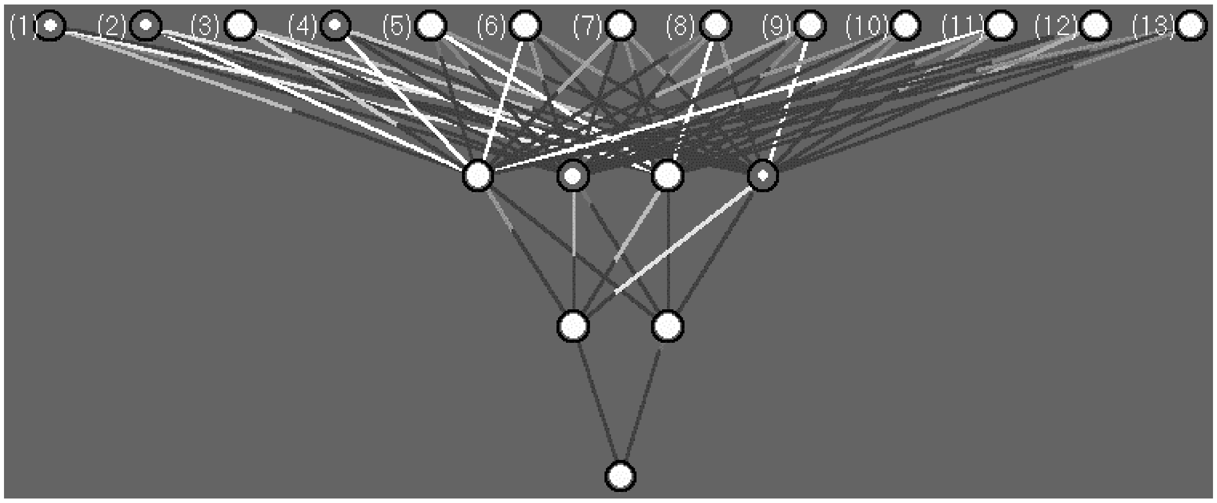

Figure 9. Network trained to predict the average market value of houses in a development.

8. LIMITATIONS

The primary limitation of this visualization tool is that the graphical feedforward network depiction we have presented does not scale well to networks with large numbers of nodes. Already in the network trained for the housing value problem (Figure 9), which has a total of 20 nodes, this problem is somewhat apparent; in larger networks it becomes even more pronounced. Additionally, the use of the sigmoid scale to display nodal activation values has made it difficult to distinguish between activations whose sigmoids are close, but which numerically are far apart (e.g. 10.0 and 20.0). The sigmoid scale was chosen due to the absence of an upper bound on nodal activations; in practice a linear or logarithmic scale with a cutoff value may be more appropriate.

9. FUTURE WORK

The problem of scalability in the graphical network visualization could be addressed via selective zooming and/or clustering of nodes. To further explore the potential of this tool as an educational device to explain aspects of evolutionary computation, a variety of evolutionary algorithms, as well as different breeding and selection schemes, could be integrated into the package. To broaden the applicability of this tool, network architectures other than feedforward could be displayed both in graphical and compact matrix form. Rigorous user evaluation should also be carried out.

10. CONCLUSIONS

We have found this visualization tool to be useful both as an educational device, to aid in the understanding of neural networks, search spaces, and genetic drift, and as a practical tool for solving complex problems with neural networks. We have been able to extract problem-specific knowledge by examining the graphical depiction of a network provided by NVIS. We have found the interactive training environment to be a great help in successfully solving problems with neural networks, and we believe that this work has relevance in other graph visualization domains, such as network traffic data. An executable demo of the NVIS system described in this paper is available at http://davis.wpi.edu/~matt/projects/NVIS/nvis.zip.

ACKNOWLEDGEMENTS

The authors thank Soraya Rana for helpful discussions and for providing a library for evolutionary strategies that was modified for use in the system described in this paper.

REFERENCES

1. K. Chellapilla, D.B. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge", *IEEE Transactions on Neural Networks*, Vol. 10, no. 6, pp. 1382 - 1391, Nov. 1999.
2. S. Makeig, T.-P. Jung, and T.J. Sejnowski, "Using feedforward neural networks to monitor alertness from changes in EEG correlation and coherence", *Advances in Neural Information Processing 8*, MIT Press, pp. 931-937, 1996.
3. S.X. Yang, M. Meng, "An efficient neural network approach to dynamic robot motion planning", *Neural Networks*, Vol. 13, No. 2, pp. 143-148, March 2000.
4. G. W. Cottrell, "Extracting features from faces using compression networks: Face, identity, emotion and gender recognition using holons", *Connection Models: Proceedings of the 1990 Summer School*, San Mateo, CA: Morgan Kaufmann, 1990.
5. R. Müller, H.-H. Hemberger, and K. Baier, "Engine control using neural networks: A new method in engine management systems", *MECCANICA*, Vol. 32, no. 5, pp. 423-430, 1997.
6. D. A. Pomerleau, "Knowledge-based training of artificial neural networks for autonomous robot driving", *Robot Learning*, pp. 19-43, Boston: Kluwer Academic Publishers, 1993.
7. A.E. Bryson, Y.-C. Ho, *Applied Optimal Control and Estimation*, Blaisdell, 1969.
8. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors", *Nature*, Vol. 323, pp. 533-536, 1986.
9. T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1995.
10. D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence,* Piscataway, NJ: IEEE Press, 1995.
11. G. Miller, P. Todd, and S. Hegde, "Designing neural networks using genetic algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379-384, 2000.
12. H-P. Schwefel, *Evolution and Optimum Searching*, Wiley Interscience, 1995.
13. M.W. Craven, J. Shavlik, "Visualizing learning and computation in artificial neural networks", *International Journal on Artificial Intelligence Tools*, Vol. 1, pp. 399-425, 1991.
14. J. Beddow, "Shape coding of multidimensional data on a microcomputer display", *IEEE Proceedings on Visualization*, pp. 238-246, 1990.
15. C.L. Blake, C.J. Merz, *UCI Repository of machine learning databases*, http://www.ics.uci.edu/~mlearn/MLRepository.html, University of California, Irvine, Dept. of Information and Computer Sciences, 1998.
16. Koza, J. R., "Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems", Stanford University Computer Science Department technical report STAN-CS-90-1314, June 1990.
17. G. Robertson, "Parallel implementation of genetic algorithms in a classifier system", *Genetic Algorithms and Simulated Annealing*, London: Pittman, 1987.
18. D. Harrison, D.L. Rubinfeld, "Hedonic prices and the demand for clean air", *Journal of Environmental Economics and Management*, Vol. 5, pp. 81-102, 1978.