N^2VIS: An Interactive Visualization Tool for Neural Networks

Matthew J. Streeter   Matthew O. Ward   Sergio A. Alvarez

Department of Computer Science
Worcester Polytechnic Institute

Abstract

We describe N^2VIS, an interactive visualization tool for feedforward neural network populations trained through evolutionary computation. N^2VIS provides visualization of network attributes including topology, connection weights, weight volatility, and nodal activation levels for specific input values, as well as of genealogical relationships between different networks. Changes to network parameters can be made interactively by the user during training, and their effects are immediately visualized. These features contribute to making N^2VIS an effective visualization tool for designers, users, and students of neural networks.

1 Introduction

Artificial neural networks (ANN) are models of adaptive distributed computation with roots in neurobiology. ANN's have been used with great success in a variety of domains to solve problems involving function approximation or classification, e.g. [8, 5, 12]. 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 ANN's 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 ANN's include error backpropagation [4, 10] and evolutionary computation [1, 11, 9]. 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.

In the present paper we describe N^2VIS, an interactive visualization tool for feedforward neural networks trained through evolutionary computation. Our work addresses the visualization of individual networks as well as that of the genealogical relationships between different networks in the population. Our experiences using N^2VIS show that it can be an effective tool for designers and users of ANN systems, as well as a valuable resource for students learning about ANN.

1.1 Relation to Previous Work

Existing techniques for visualization of neural network weights include Hinton diagrams and Bond diagrams, e.g. [6]. Hinton diagrams provide a box-like display of network weights that does not reveal network topology. Bond diagrams superimpose a visual depiction of connection weights on a planar projection of the network graph, thus conveying topological information. However, neither of these approaches includes a mechanism for the visualization of nodal activation values. N^2VIS, the tool described in the present paper, displays network topology and connection weights, as well as nodal activation levels for specific values of the input signals. A key difference from preceding work is that we address the visualization of family relationships arising from the evolutionary adaptation process. The volatility of the weight of a given connection, a measure of the change in the value of the weight with respect to its value in the previous generation, is also displayed visually. Furthermore, N^2VIS allows the user to interactively adjust training parameters during adaptation; the results of this interaction are 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

Figure 1 depicts a feedforward neural network drawn from a population trained to determine the parity of a four bit number. The network is presented both as a weighted graph and in compact matrix form.

The 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.

In this representation, the diameter of the white circles inside each node has been mapped on a sigmoid scale to the node's activity level. 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 (see color plate). 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.

The visualization 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 refine a network via backpropagation, and to manually adjust the weights using each colored bar as a slider.

3 Compact Matrix Representation

The three colored matrices depicted in the lower right corner of Figure 1 represent an enlarged version of the compact matrix representation for the network discussed in the previous section. 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 (see color plate). This representation has the convenient property that the

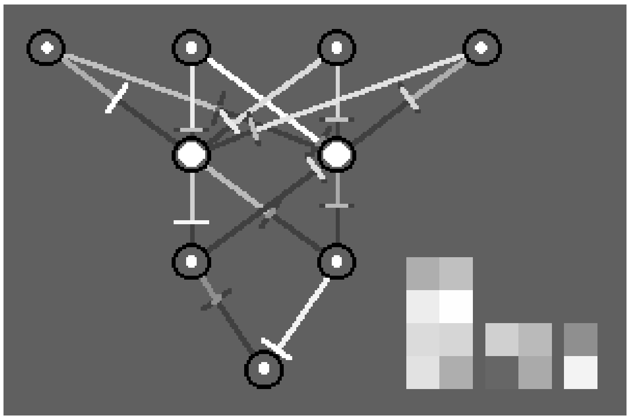

Figure 1: Network trained to determine the parity of a four bit number

multiplication of the matrices in the compact representation yields the final transformation matrix for a network using the linear activation function F(x)=x. We note that the compact matrix representation employed here bears a superficial resemblance to glyphs as previously used to visualize multidimensional data sets [2]; however, in the present case the representation is more directly related to the known structure of the networks being considered.

4 Heredity Visualization

The purpose of the compact matrix representation is to allow many networks to be displayed on the screen at once. In Figures 2 and 3, we present the output of two visualizations designed to take advantage of this: the "Generations" window (Figure 2), which organizes all the networks in the population history by generation, and the "Family Tree" window (Figure 3), which depicts the family tree for a specific network.

Figure 2 illustrates the first six generations of a pool of networks trained to predict the performance of a CPU based on six inputs, including cache size, main memory size, and MHz rating (data taken from [3]). 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. The green lines in this visualization denote parentage; white lines indicate survival from one generation into the next. There are no white lines in this particular illustration (see color plate).

Figure 3 illustrates the family tree for the network selected in figure 2. 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.

5 Results

Our experiments have shown this visualization tool to be of use in four areas: manually designing networks to solve problems, determining the shape of the error surface for a particular problem and topology, understanding the phenomenon in evolutionary computation known as genetic drift, and extracting domain knowledge from the operation of a network. Each of these areas is elaborated upon below.

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 or to refine an existing one. 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.

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. In addition to the "Network Editor" "Family Tree" and "Generations" windows described above, this tool provides a simple 2D plot of fitness vs. time, making it easier for the user to gauge the progress of the evolutionary algorithm. In Figure 6, for example (see color plate), we observe the existence of several plateaus in the error surface, 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.

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 2, in which, after only six generations, all of the networks in the population can be seen to be very similar. In Figure 2, it can readily be seen that the

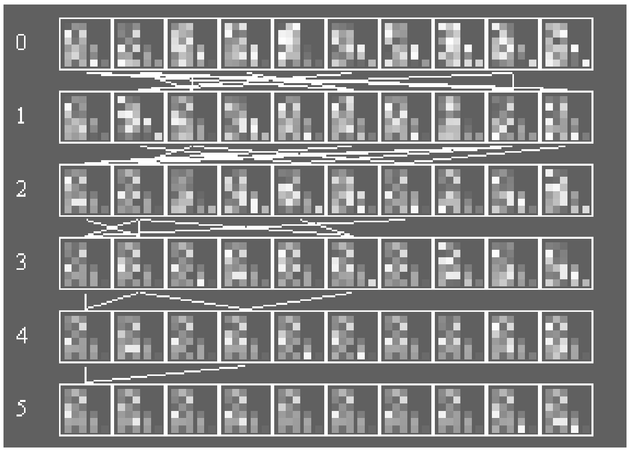

Figure 2: First five generations of a pool of networks trained to predict CPU performance

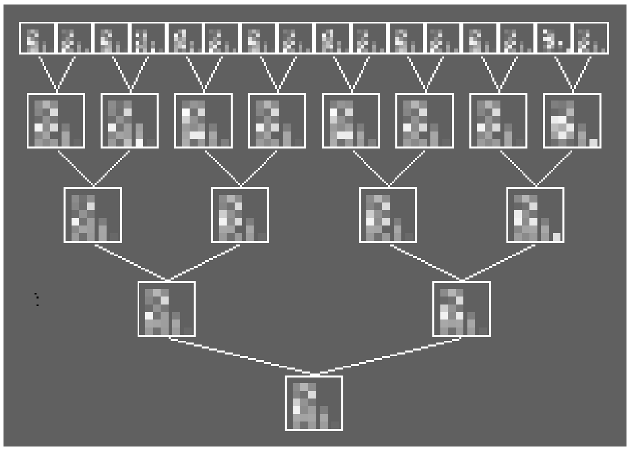

Figure 3: Family tree for selected (lower left) network in Figure 2

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 8 on the color plate presents a network trained to predict the average market value of houses in a development based on 13 inputs, including property tax rate, student/teacher ratios, and distance to major employment centers [3, 7]. 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 see 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, tells us that houses with more rooms tend to be worth more. In principle, this type of analysis could be applied to a problem domain for which no a priori knowledge exists.

6 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 8, color plate) 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.

7 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 evolutionary computation, a variety of evolutionary algorithms (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 evaluation should also be carried out.

8 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 N^2VIS. 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 N^2VIS system described in this paper is available at http://www.wpi.edu/~mjs/mqp/notes/NNVis.zip.

Acknowledgements

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

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice*, Oxford U. Press, 1995.

[2] J. Beddow. Shape coding of multidimensional data on a microcomputer display. *Proc. Visualization '90*, IEEE Computer Society Press, pages 238-246, 1990.

[3] 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.

[4] A.E. Bryson, Y.-C. Ho. *Applied Optimal Control and Estimation.* Blaisdell, 1969.

[5] 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, pages 1382 - 1391, Nov. 1999.

[6] M.W. Craven, J. Shavlik. Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, Vol. 1, pages 399-425, 1991.

[7] D. Harrison, D.L. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, Vol. 5, pages 81-102, 1978.

[8] S. Makeig, T.-P. Jung, T.J. Sejnowski. Using Feedforward Neural Networks to Monitor Alertness from Changes in EEG Correlation and Coherence. in *Advances in Neural Information Processing 8*, MIT Press, pages 931-937, 1996.

[9] G. Miller, P. Todd, S. Hegde. Designing Neural Networks Using Genetic Algorithms. *Proc. Third International Conference on Genetic Algorithms*, pages 379-384, 2000.

[10] D.E. Rumelhart, G.E. Hinton, R.J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, Vol. 323, pages 533-536, 1986.

[11] H-P. Schwefel. *Evolution and Optimum Searching.* Wiley Interscience, 1995.

[12] S.X. Yang, M. Meng. An efficient neural network approach to dynamic robot motion planning. *Neural Networks*, Vol. 13, No. 2, pages 143-148, March 2000.

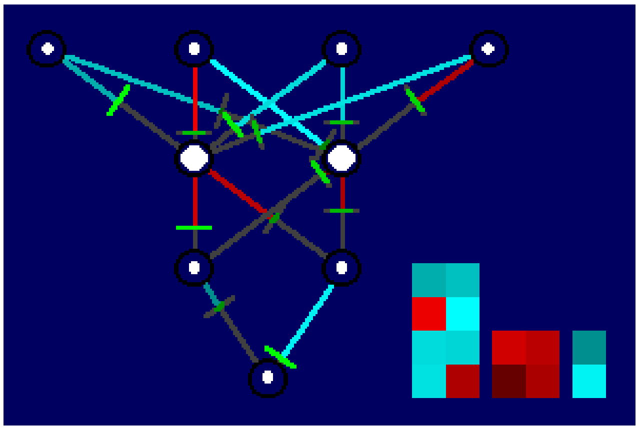

Figure 4. Network trained to determine the parity of a four bit number

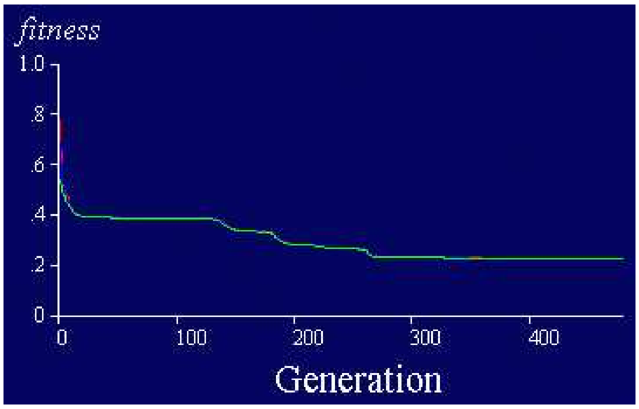

Figure 5. Fitness plot for pool of networks trained to test for winning position on a tic-tac-toe board

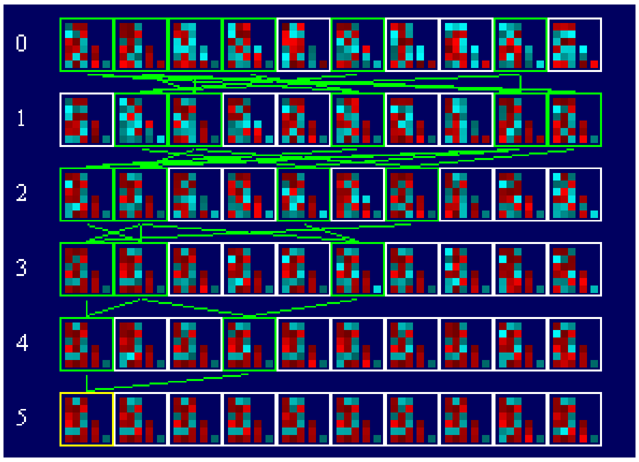

Figure 6. First five generations of a pool of networks trained to predict CPU performance

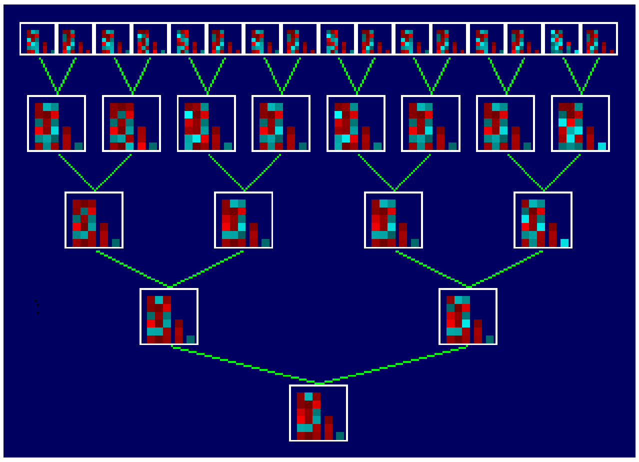

Figure 7. Family tree for selected (lower left) network in Figure 6

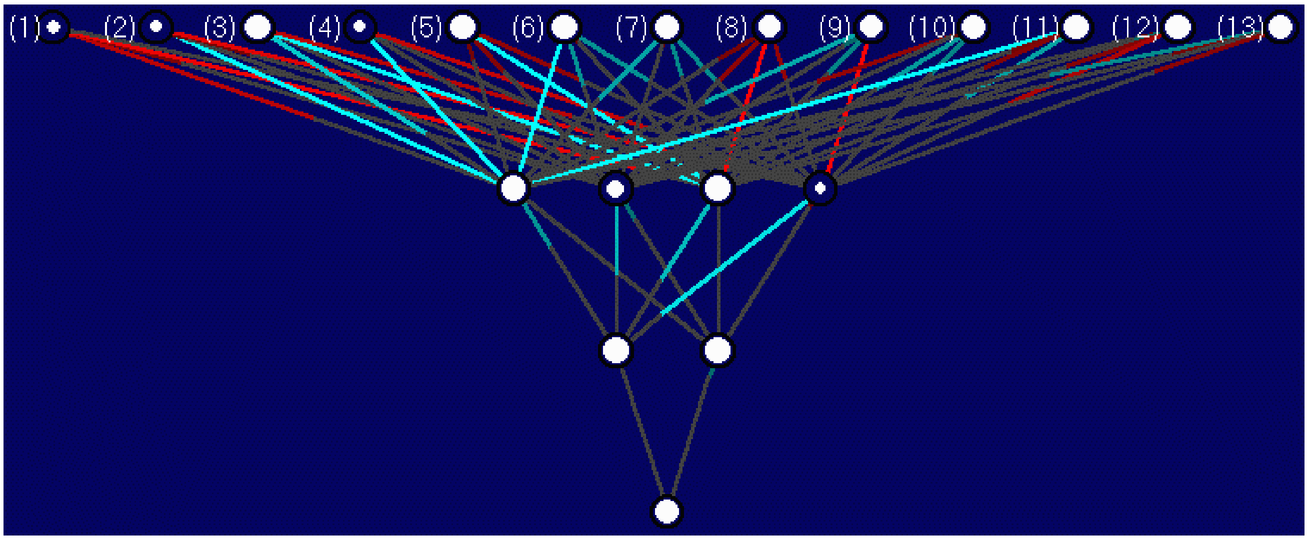

Figure 8. Network trained to predict the average market value of houses in a development