Visualization of neural networks using Java applets

I. FISCHER, A. ZELL

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen, Köstlinstr. 6, 72074 Tübingen, Germany
tel: +49 7071 2977176. fax: +49 7071 922983. e-mail: {fischer, zell}@informatik.uni-tuebingen.de

Abstract

During the last decade, artificial neural networks (ANNs) have reached maturity and established themselves as a useful tool for information processing, especially for complex data, where prior knowledge and models are limited. Unfortunately, ANNs are themselves also complex, usually consisting of many interconnected non-linear processing units (neurons), what makes them hard to inspect and analyze. As a part of Baden-Württemberg state project *VirtuGrade*, we have developed a number of interactive visualization tools for educational purposes. They help users to gain insight of what is happening inside a neural network and improve their understanding of the subject. As a platform, we use Java applets, embedded in an on-line book covering the topic.

INTRODUCTION

Artificial neural networks (ANNs), or simply neural networks, have been developed since the mid-fifties, following theoretical foundations laid already in the early forties. Inspired by their biological counterparts, artificial neural networks consist of a large number of highly interconnected processing units ("neurons"), whose connections resemble synapses in biological networks. The development of artificial neural networks has been motivated by some attractive properties observed in biological networks, most importantly the ability to learn from examples, ability to generalize knowledge, ability to extract prototypes from noisy data, high fault tolerance and inherently parallel processing.

Today, neural networks are recognized tools for data analysis, when the data is complex and the knowledge about it is sparse. Contrary to statistical methods, neural networks don't rely on a specific model for data being analyzed, although they also make certain weak assumptions.

In most types of neural networks, especially multilayer perceptrons, the knowledge is distributed: one can't localize a distinct part of it which is responsible for certain behavior. Instead, the whole network participates in actions. Although much of the mentioned attractivity of neural networks is a direct consequence of that fact, it is also their major drawback: it is extremely hard to analyze the knowledge stored in a network and to gain insight what is happening in it and why. High interconnectivity results in high dimensionality of parameter space, and compounded with non-linearities involved in each neuron, it makes mathematical analysis very difficult. In other words, it is very hard to understand a neural network, to describe its knowledge in simple terms.

This problem is further accentuated in education, if one tries to teach someone about neural networks. It is easy to understand that students have problems in imagining relationships in high-dimensional vector spaces.

To counter this problem, we have developed a set of interactive visualization tools, in order to give students insight at least in some simple, but nevertheless fundamental principles of the topic. The tools are implemented in Java and embedded as applets in an on-line version of the textbook used in lectures. Java was chosen as platform for the following reasons:

1. It is a powerful, easily extensible object-oriented language that includes all features needed, from mathematics to graphics.

2. It is available for all major operating systems.
3. Most users have it already installed, as soon as they install a web browser.
4. Java programs work over the network and no local installation is needed.
5. Java programs can be easily embedded into web pages.

MOTIVATION

Let us consider a typical network, as shown in figure 1. The circles represent neurons and the arrows connections between them. The network is an ordinary feed-forward network, also called multilayer perceptron (MLP). It consists of neurons organized in layers: an input layer, a hidden layer (a layer that is not directly visible, neither from the input nor from the output side) and an output layer. Each layer is connected with the next one through a set of links, each having a specific and generally adjustable “weight”, which is simply a multiplicative factor.

Neurons in the input layer serve only as interface to the outer world: they pass input data to the rest of the network without any processing. For that reason, the input layer is usually not counted when one talks about number of layers in a network. In neural network terminology, the shown network would be commonly referred to as a two-layer network. Another rationale is that it's the connections that can be adjusted to achieve certain behavior, and this network has two layers of them.

On the way from the input to the hidden layer, signals are amplified by the weights of corresponding connections. Neurons in the hidden layer sum all the signals that reach them and usually add an internal value, the *bias*, to the sum. Finally, they process the result. In this very simple example, we shall suppose that the processing consists solely of passing the signal through an element with a nonlinear transfer function. The usual choice for it is *logistic function*:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

or one of its modifications. A schematic representation of the neuron is shown in figure 1 right.

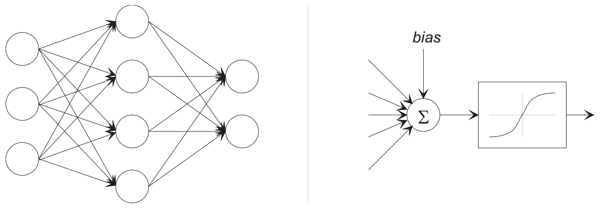

Figure 1: A simple neural network (l.) and neuron

For most neural network models it is assumed that all neurons in a network, or at least in a layer, are of the same type (the input layer not counted). This significantly simplifies the model without generally reducing its capabilities.

In our network, the neurons in the output layer do basically the same as the hidden neurons. The only difference is that they don't pass the processed signals anywhere further.

The described network actually defines a function $\boldsymbol{R}^3 \rightarrow \boldsymbol{R}^2$. Its shape can be influenced by adjusting the connection weights and neuron biases – altogether 26 parameters. However, the graphical representation of the function is impossible, because already the input space exhausts our three real-world dimensions. And the nonlinearities in each neuron make it hard to analytically inspect it. So even this simple network might be too complex for a student to understand.

COMMUNICATING THE BASICS

As we see, we have to start with simpler examples. In the applets that accompany our courses, we use fully functional neural network algorithms, but the networks themselves and their visualization are optimized to reflect only important properties and hide unnecessary details. The user interface is tuned for education purposes and not for industrial or scientific applications.

1. Linear separability

Figure 2 shows a screenshot of an applet with a minimalistic network: two input and one output neuron. The output neuron has step (or threshold) transfer function:

$$f(x) = \begin{cases} 0 & \text{for } x \le 0 \\ 1 & \text{for } x > 0 \end{cases} \qquad (2)$$

On the right of the network is the visualization of the function it implements. This applet introduces the concept of linear separability, which is one of the key concepts in neural networks.

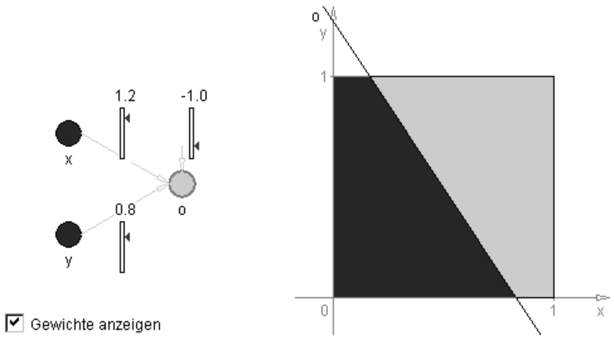

Figure 2: Linear separability applet

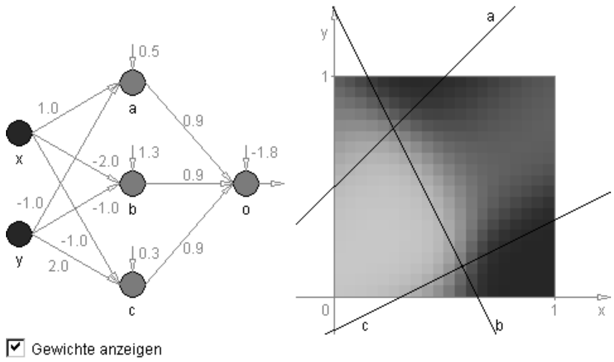

Figure 3: Perceptron applet

The network divides two-dimensional input space, determined by the two input neurons, into two half-spaces, separated by a line. In a more general case, having *n* input neurons, an *n*-dimensional space would be divided into two half-spaces by a *n-1*-dimensional hyperplane. All input data – i.e, all combinations of *x* and *y* – can be classified as belonging to one or another half-space. Now, the student can use sliders to adjust weights and bias in the network and observe their influence to the space division.

2. Perceptron

A slightly more complicated network is shown in figure 3. It is a two-layer perceptron with logistic transfer function of the neurons. Perceptrons are actually a whole class of neural networks which has been extensively investigated since the mid-fifties [6], and still belong to the most important neural network architectures.

As in the previous applet, the user can interactively explore how different weights and biases influence the perceptrons output function. This perceptron can approximate a certain subclass of $[0, 1]^2 \rightarrow [0, 1]$ functions, and the output is chroma-coded: blue (dark) corresponds to 0 and green (light) to 1. The most common application of perceptrons like this one is pattern recognition. The network output can be interpreted as follows: 1 means that a pattern has been recognized to belong to a certain class and 0 not. Since the values can be non-integer (as can be seen near the separating lines in the display area), a more correct interpretation is that the output value represents the confidence level that the pattern belongs to a class.

The transition from 1 to 0 near the separating lines is in reality smooth. However, for better responsiveness, the applet computes the output value in reduced number of discrete points, which results in shown graininess of the image.

3. Backpropagation

A further extension to the applet introduces the concept of learning. It is probably the most important concept in neural networks. Adjusting weights manually to achieve desired behavior of a network is cumbersome even for the presented toy networks. For serious applications, involving hundreds of neurons with many thousands connections, it is virtually impossible. Therefore, algorithms for automatic adjustment of weights have been developed. The process is called learning, since the network changes its behavior based on examples.

Certainly the most famous learning algorithm is backpropagation [7], [8]. Although in the meantime more efficient procedures have been developed, the importance of backpropagation, conceptual as well as historical, remains unchanged.

Our backpropagation applet has an appearance similar to the previous two. In addition, the user can select desired network response in four vertices – (0, 0) to (1, 1) – of the input space and then observe how the network “learns”' the patterns, i.e. how its function changes as it approaches the target function.

SOME SIMPLE EXAMPLES

1. Function approximation

The capability of neural networks to approximate real-valued functions is visualized by another applet, shown in figure 4. In this applet, the network, shown in the lower part of

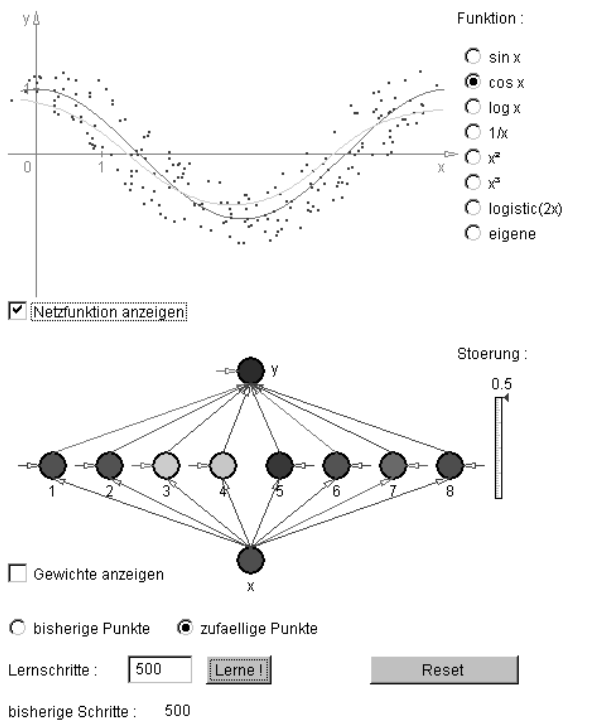

Figure 4: function approximation applet

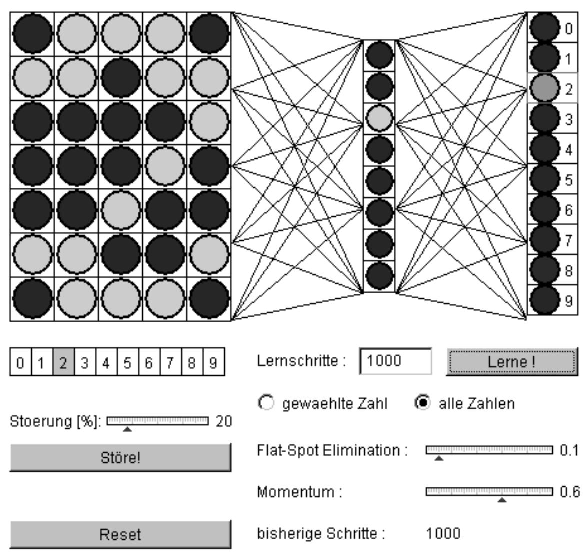

Figure 5: Pattern recognition applet

the display, can approximate a function $\boldsymbol{R} \to \boldsymbol{R}$, which is drawn in the graphic area above it. The user can choose one of the predefined functions or define his own by entering points in the function display area. The approximate function that the network implements can be shown in a different color.

Another interesting property of neural networks, the ability to extract prototypes from noisy data, can also be observed with this applet. By adding uniform noise to the data, the user can see how disturbances affect the learning process and its outcome.

2. Pattern recognition

Another big application field for neural networks is pattern recognition. The ability of neural networks to learn from examples, even from noisy ones, to extract the representative patterns (prototypes) and to make generalizations and apply them to never seen data makes them very interesting for that purpose. However, there are limitations to what they can, which shouldn't be overlooked.

In order to give students a feeling how the pattern recognition works, we have developed an applet that simulates a network for recognizing the ten digits, as being displayed in a common 5×7 ASCII matrix. In this applet, shown in figure 5, the input neurons of the network are organized in such a matrix and the output layer consists of ten neurons, each representing a digit. The activity of each output neuron reflects the network's confidence that the pattern on the input represents the corresponding digit.

The network can be trained with the digits (all of them or only some), optionally with added noise. The response of the trained network can then be tested using again ideal or noisy digits. A well-trained network will be able to generalize and even to correctly classify never seen patterns, like those manually entered by the user.

The hidden layer of the network is of such size that the user easily notices one weakness of backpropagation: the algorithm is rather slow, i.e. it takes a rather long time to train the network. To overcome that problem, he or she can use two additional parameters, the so called *momentum term* and *flat spot elimination*. These are two earliest improvements to the backpropagation algorithm that are still commonly used.

DYNAMICS IN NEURAL NETWORKS

Until now we have been dealing only with feed-forward networks. Still, there exist some other important architectures that we would like to teach our students about.

1. Learning vector quantization

Learning vector quantization (LVQ) [4] is an astonishingly simple yet efficient concept: to each pattern class, a number of codebook vectors is assigned and initially dispersed through the input space. During training, they get

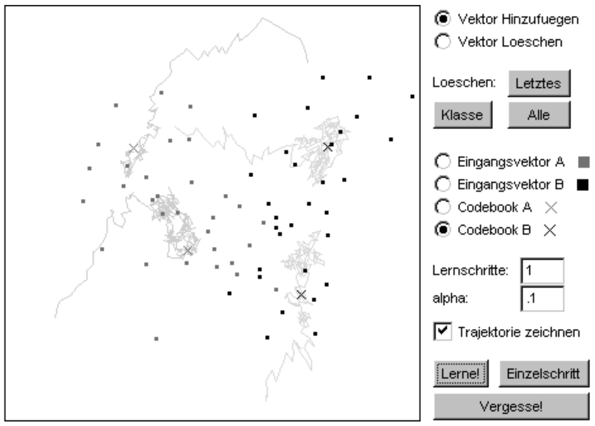

Figure 6: Learning vector quantization applet

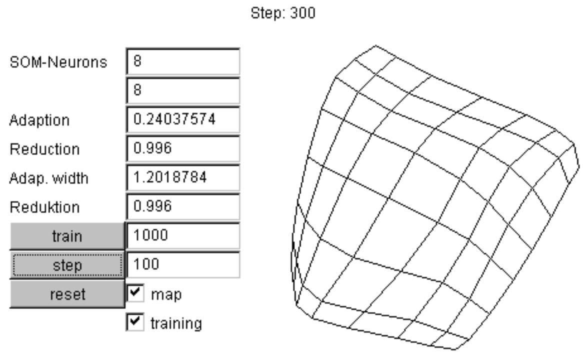

Figure 7: Self-organizing map applet

attracted by input vectors of their own class and rejected by vectors of all other classes. Finally, after enough training steps, the codebook vectors divide the space among themselves according to the distribution of input data.

An applet that visualizes that behavior is shown in figure 6. The left part of the screen shows the two-dimensional vector space, where the user can enter input vectors (marked as dots) and initial positions of codebook vectors (shown as crosses). On the right are controls where he or she can set some learning parameters and initiate the learning process. During training, the codebook vectors move through the vector space. Their trajectories, which seem chaotic (they are actually non-deterministic, since the input vectors are chosen randomly) can also be displayed or hidden, for maintaining better oversight.

2. Self-organizing maps

A somewhat similar network architecture are self-organizing maps [5], also referred to as Kohonen maps. They are usually used for mapping complex, multidimensional numerical data onto a geometrical structure of lower dimensionality, like a rectangular or hexagonal two-dimensional lattice. The mappings are useful for visualization of data, since they reflect the similarities and vector distribution of the data in the input space. Each node in the map has a reference vector assigned to it. Its value is a weighted average of all the input vectors that are similar to it and to the reference vectors of the nodes from its topological neighborhood.

Self-organizing maps are one of the most popular neural network architectures. Between their introduction and 1995 there have been over 1500 publications covering them [3]. In the applet that we use for their visualization (figure 7), a two dimensional map spans a two dimensional input space, where the input data is uniformly distributed inside a square. Other distributions, like triangle or circle, are also planned. The unfolding of the map is an iterative process and its visualization leads to an interactive animation.

3. Hopfield networks

Hopfield networks [1] are another architecture that has significantly influenced the renaissance of neural networks in the early eighties. Contrary to the previously presented models, the Hopfield networks are recurrent i.e. the network output is fed again to the input. They are characterized by an interesting relaxation dynamics which leads them to stable states called *energy minima*, in analogy to physical energy states in crystals.

In their well known paper [2], Hopfield and Tank reported that Hopfield networks are capable of solving the *traveling salesman problem* (TSP). The figure 8 shows an applet that demonstrates this. The user can place “towns” in the two-dimensional input area on the left and enter network parameters in the input fields. After the network stabilizes, the result is shown graphically in the input area, as lines connecting the towns on the path, as well as coded in the Hopfield matrix (the checkered field on the right).

EDUCATIONAL EMBEDDING

The applets presented above are used in lectures on neural networks, held at the University of Tübingen. Technically, they are embedded in the on-line (HTML) version of the textbook “Simulation Neuronaler Netze” [10], where

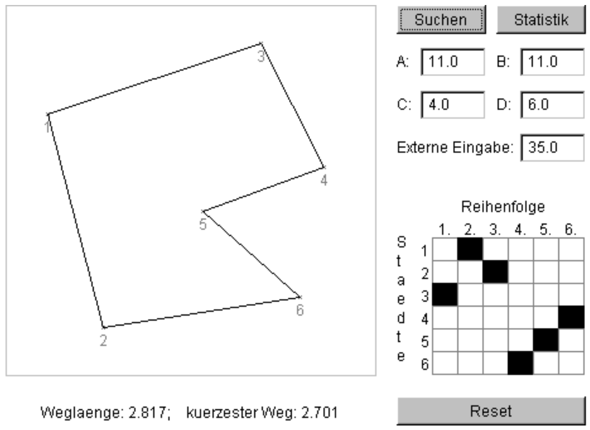

Figure 8: Hopfield network for TSP applet

they are used instead or in addition to static images at places where interactivity is preferred.

The book accompanies the lectures. The on-line version, whose first version has been available since 1998, is especially intended for self-education and as a reference and, according to our experience, it is mostly used during exercises. The evaluation of server log files reveals that pages containing the applets are much more attractive than the rest of the book. The students visit them more often than static pages and remain longer occupied with them.

The applets are well suited for visualization, but are not capable of solving more complex problems. Therefore, in the exercises we mostly use the *SNNS* [9], [11], an industrial-strength neural network simulator. A closer connection between it and the applets is planned, and we plan to further extend their number in the on-line book. The book is also intended for use for continuing education and professional training in industry. It is being developed in scope of the it *VirtuGrade* project, which is itself one of the *Virtual University* projects sponsored by the state Baden-Württemberg. The applets are currently available only to our students, although it is being considered to make them public in the future.

CONCLUSION

Although neural networks can be a powerful tool for information processing, they are also complex and somewhat hard to understand. To facilitate education in neural networks, we have developed a new set of interactive visualization tools. They are implemented in Java and consist of full-featured neural network simulation routines and a graphical user interface, and are intended to give inexperienced users a better insight into abstract processes, interdependencies and dynamics of neural networks. Results in the first year of their availability to students justify our expectations.

ACKNOWLEDGMENTS

Most of the presented applets have been programmed by Igor Fischer and Fabian Hennecke. The applet for visualizing self-organizing maps has been developed by Josef Göppert. The *SNNS* was developed by a team around Andreas Zell and is being maintained in Tübingen.

References

[1] J.J. HOPFIELD, “Neural Networks and physical systems with emergent collective computational abilities”, Proc. National Academy of Sciences, USA, Vol. 79, pp. 2554-2558

[2] J.J. HOPFIELD, D.W. TANK, “’Neural’ computation of decisions in optimization problems” Biological Cybernetics 52 (1985) 141-152

[3] T. KOHONEN “Self-Organizing Maps”, Springer Series in Information Sciences, vol. 30, Springer Berlin Heidelberg 1995.

[4] T. KOHONEN, “Improved versions of learning vector quantization”, Proc. IJCNN, San Diego (1990) 545-550

[5] T. KOHONEN “Self-Organized formation of topologically correct feature maps”, Biological Cybernetics 43 (1982) 59-69

[6] F. ROSENBLATT, “The perceptron: a probabilistic model for information storage and organization in the brain”, Psychological Review 65 (1958) 386-408

[7] D.E. RUMELHART, G.E.HINTON, R.J.WILLIAMS, “Learning representations by back-propagating errors”, Nature 323 (1986) 533-536

[8] P.J. WERBOS, Backpropagation: Past and future, Proc. ICNN, I, (343-353) IEEE Press, New York, 1988

[9] A. ZELL *et al.* “SNNS User Manual, Version 4.1”, University of Stuttgart, IPVR, Report Nr. 6/95, 1995

[10]A. ZELLl, “Simulation Neuronaler Netze”, Addison-Wesley (Deutschland), 1994

[11]A. Zell *et al.* “SNNS (Stuttgart Neural Network Simulator)” book chapter in: J. Skrzypek, Neural Network Simulation Environments, Kluwer Academic Publishers 1994