# Neural Networks and Orbit control in Accelerators*

Eva Bozoki and Aharon Friedman
National Synchrotron Light Source, Brookhaven National Laboratory, Upton, NY 11973

**Abstract**

An overwiew of the architecture, workings and training of Neural Networks is given. We stress the aspects which are important for the use of Neural Networks for orbit control in accelerators and storage rings, especially its ability to cope with the nonlinear behavior of the orbit response to 'kicks' and the slow drift in the orbit response during long-term operation. Results obtained for the two NSLS storage rings with several network architectures and various training methods for each architecture are given.

## 1 INTRODUCTION – WHY NNETS?

Neural Networks, Nnet's for short, are radically different from standard software methods [1]. The outcome of NNet's does not depend on the programmer's prior knowledge of rules; Nnet's can learn underlying relationships even if they are hard to find and can solve problems that lack existing solutions. Nnet's can also generalize, they are able to correctly process imperfect or incomplete or noisy data. Nnet's are ideally suited to handle nonlinear problems, since there are more parameters to adjust then the total number of variables. Since the Nnet's can continuously learn from it's own mistakes, it can be also used when the system's behavior is drifting in time.

All these features makes Nnet's highly suitable for orbit correction in accelerators and storage rings, where the actual relationship between measured orbits (input set) and the required orbit correction (output set) is nonlinear due to the presence of sextupoles and other nonlinear elements, and where the orbit response of the system is not constant in time.

## 2 ARCHITECTURE

Neural Networks consist of nodes and weighted and directed connections between them. The connections can be feedforward, feedback, recurrent or any combination of those. Network can also differ in the way they are trained to solve a problem; supervised learning and self teaching networks [2].

In a feedforward network, what we will concentrate on in this paper, the nodes are grouped in layers; there is one input-layer, one output-layer and any number of hidden-layers. [1] Each nodes in the input-layer receive one (environmental) input signal, any other nodes can have arbitrary number of inputs (which are the output of other nodes). All nodes have one output signal, and any node can have a threshold. The processing of the input signals to generate the output signal is taking place in the nodes. Fig.1 shows a typical node, where $o_i$ is the output signal of the i-th node with $w_{ij}$ synoptic weight between the i-th and the j-th nodes, $\eta_j$ is the threshold and the processing is represented by the $f_j$ function. Fig. 2 shows three different feedforward networks: (a) fully connected, when each node of one layer is connected to each node in the next layer, (b) shortcut connected, when all nodes are connected to each other and (c) cascade network, which will be discussed in Section 4.1.

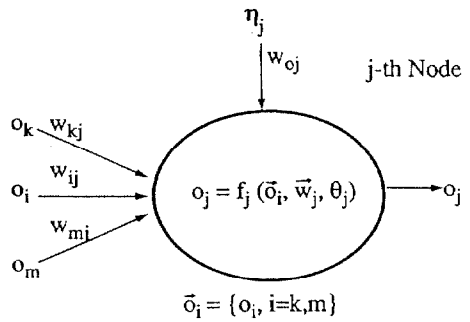


Fig. 1: Schematic representation of a Node

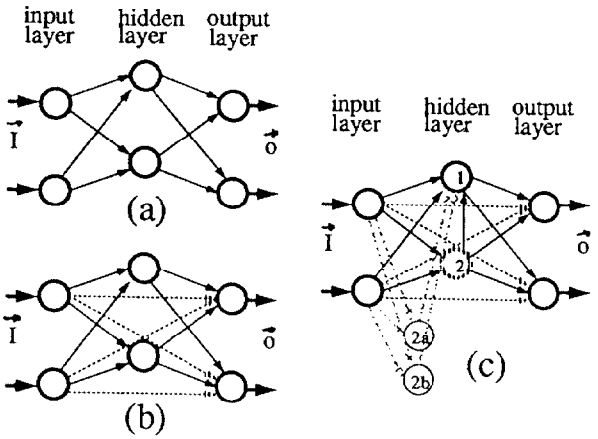


Fig. 2: Network connections: (a) Fully connected, (b) Shorcut connected and (c) Cascade network.

When a set of input signals, $\vec{I}$, is presented to the Nnet, then the output of each nodes propagates to every other

*Work performed under the auspices of the U.S. Dept. of Energy under contract no. DE-AC02-76CH00016.

[1] Hidden layers and/or recurrent connections enable the Nnets to handle nonlinear problems — A two layer feedforward network would not be able to handle nonlinear problems

node it is connected to and this output is modified by the synoptic weight characterizing the connection between those nodes. At the end of this propagation process a set of output signals, $\vec{o}$, is emerging from the Nnet.

## 3 PROCESSING

In general terms, the following processing may take place in a node (see Fig. 3). All the $o_i, i = k, m$ inputs to the j-th node are processed by a *transfer function* to produce the net input, $\tilde{i}_j$. In most cases the transfer function is the weighted sum (or product) of the individual inputs [2] :

$$\tilde{i}_j = f_j^{tr}(o_i, w_{ij}) = \sum_i w_{ij} o_i \quad .$$

The net input is then processed by the *activation function* to produce the $a_j$ activation function of the node:

$$a_j = f_j^{act}(\tilde{i}_j, \eta_j),$$

where $\eta_j$ is a possibly non-zero threshold value (bias). There is an almost endless choice for activation value, the most widely used being the step function, ramp function, sigmoid function and tanh function, as illustrated on Figs.4.

The last step in the processing is provided by the *output function*, which in most cases is identity or possibly clip-function:

$$o_j = f_j^{out}(a_j) = a_j \quad or \quad = \begin{cases} a_{min} - & if \ \ a_j \le a_{min} \\ a_j \ \ - & a_{min} < a_j < a_{max} \\ a_{max} - & a_j \ge a_{max} \end{cases}$$

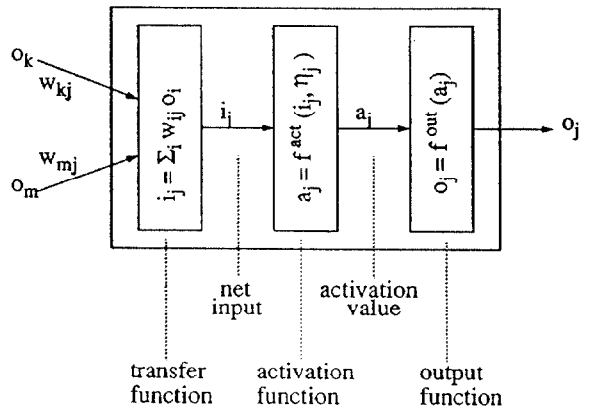


Fig. 3: Processing by a Node.

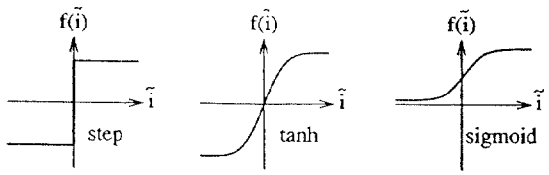


Fig. 4: Three frequently used activation functions

[2] Input nodes receive one input from the environment. Thus for input nodes the synoptic weight is 1 and the transfer function is identity

## 4 LEARNING

Once the architecture, including the connections is established, the synoptic weights have to be calculated/adjusted to ensure that the Nnet will produce the right output-set for any given input-set. This is the learning phase. The two basic categories of learning are the *supervised* and the *self teaching* methods. In the former, training sets (consisting of $\vec{I}$ inputs and the corresponding desired $\vec{O}$ outputs) are presented to the Nnet and the learning continues until the difference between the $\vec{o}$ actual and the $\vec{O}$ desired output of the Nnet is less then the desired accuracy of the system. In our application, we will compare such supervised learning methods.

In the *unsupervised* method the learning is by competition among neighbor nodes without any 'teacher'. This category includes the Hebbian and the Competitive - Cooperative methods, like the Grossberg and ART methods, to mention only a few. Unsupervised learning is mainly used for classification and pattern recognition problems.

### 4.1 Strategies for supervised learning

Presenting an $\vec{I}$ input set to an untrained Nnet will produce after the first forward pass an $\epsilon_j = f(o_j - O_j)$ error in the j-th output node. In the *error correction learning* methods, the synoptic weights are adjusted at each iteration according to the $\vec{\epsilon}$ errors in the output nodes: $\delta w_{ij} \approx o_i \epsilon_j$. In the *reinforcement learning* methods a scalar $\epsilon_{out}$ error is calculated and the synoptic weights at each iteration are adjusted as $\delta w_{ij} \approx \epsilon_{out}$. In the *stochastic learning* methods the $\delta w_{ij}$'s are chosen randomly and the ones, which minimize the 'energy' of the Nnet, are chosen at each iteration.

A few of the error correction methods, the ones we are using, are highlighted in the followings. [3]

*Vanilla Backpropagation*

This method is searching for the minimum in the error surface by a generalized delta-rule. The synoptic weights are adjusted at each iterative step as

$$\Delta w_{ij} = \eta o_i \delta_j = \eta o_i f'_j \epsilon_j;$$

$$where \ \ \delta_j = f'_j(i_j) \begin{cases} (O_j - o_j) & if \ j \ is \ output \ node \\ \sum_k \delta_k w_{jk} & if \ j \ is \ hidden \ node \end{cases}$$

*Enhanced Backpropagation*

In this method a momentum term, $\alpha$, and a flat spot elimination term, c, are used when searching for the minimum in the error surface:

$$\Delta w_{ij}^{t+1} = \eta o_i \delta_j + \alpha \Delta w_{ij}^t,$$

$$where \ \ \delta_j = [f'_j(i_j) + c] \begin{cases} (O_j - o_j) & if \ j \ is \ output \ node \\ \sum_k \delta_k w_{jk} & if \ j \ is \ hidden \ node \end{cases}$$

*Quickprop*

This method uses information on the curvature of the error

[3] Details on these methods can be found in [3]

surface from the 2nd order derivatives and takes a direct step to the error minimum:

$$\delta w_{ij}^{t+1} = \frac{S^{(t+1)}}{S^{(t)} - S^{(t+1)}} \delta w_{ij}^{t} \quad where \quad S = \frac{\partial E}{\partial w_{ij}} \quad .$$

*Backpercolation*
In this algorithm the $w_i j$ synoptic weights are changed according to the $\epsilon_j$ node error and not according to the error in the output layer.

*Cascade*
It can find the optimal number of hidden nodes needed for a task. It starts with a network containing input and output layer only, then minimizing the overall error of the Nnet, it adds hidden nodes one by one from a pool of candidate nodes (see Fig. 2c). The actual training algorithm can be any of the previously discussed.

## 5 ORBIT CORRECTION WITH NNET

In a circular accelerator or storage ring, the relationship between the $\Theta_j$ kick introduced at the $j$-th corrector and the $X_i$ orbit change observed at the $i$-th orbit monitor is given by the following equations [4]

$$X_i = A_{ij}\Theta_j \quad , i = 1, N_m \quad , \qquad j = 1, N_c \qquad (1)$$

where $N_m$ and $N_c$ are the number of monitors and correctors in the machine and $A$ is the Response Matrix.

It is easy to see how Nnet's lend themselves for orbit correction. The environmental input of the Nnet are the measured orbit positions, $\vec{X} = \{X_i, \quad i = 1, N_{mon}\}$ and the output of the Nnet is the $\vec{\Theta} = \{\Theta_j, \quad j = 1, N_{cor}\}$ corrector kicks. For each monitor or corrector there is one input or output node, respectively. The hidden layer will make the Nnet nonlinear, and thus enable us to treat the nonlinearities, due to the presence of nonlinear elements in the machine, in the orbit response. It is also possible to include nonlinearities arising from nonlinear horizontal-vertical coupling in the measurement of the orbit simply by treating the horizontal and vertical case simultaneously, in which case the number of input and output nodes are equal to the number of horizontal and vertical elements in the monitor - corrector system.

Training sets can be generated either by calculating $\vec{X}$ for a given $\vec{\Theta}$ using the calculated or measured Response Matrix as in Eq. (1), or by actually measuring the $\vec{X}$ orbit response to a $\vec{\Theta}$ kick. One have to be careful in chosing the training sets to (i) include random linear combinations of eigen kick vectors [4], (ii) have a good balance between positive and negative decomposition coefficients to prevent an Nnet which is biased in one direction, (iii) choose small enough kicks to avoid going too deep into the non-linear regime of the activation function and to avoid beam loss but large enough to produce noticeable orbit change.

Since the Nnet output have to be symmetric around zero, hyperbolic tangent as the activation function is a good choice. The linear regime of this function is limited to $|Tanh\ x| \leq 0.1$.

The Nnet is then trained to a desired accuracy, and is ready to perform on-line orbit correction. However, the (re)training continues, since every time when the Nnet is performing an orbit correction, the error is feed back. This retraining algorithm provides fine tuning of the synoptic weights as well as the ability to adopt to any change in the orbit response.

## 6 FINDINGS

The authors in [5] compared five different Nnet architectures (2 and 3 layer fully and shortcut connected Nnets with different numbers of hidden nodes and a cascade network) and four different training methods (Enhanced backprop, Quickprop, Backpercolation and Cascade with Quickprop). For this test 200 training patterns were used for the NSLS's VUV storage ring, which has 24 orbit monitors and 16 correctors. The $\vec{X}$ inputs were calculated from Eq. (1) for the 200 random $\vec{\Theta}$ outputs. They arrived at the following conclusions: (i) a 'fully connected' three layer network is not trainable with any of the teaching methods, (ii) all but the Cascade network is trainable to similar accuracy and with similar number of iteration cycles when they are taught with one of the methods and all teaching methods are training the non-cascade type networks to similar accuracy and with similar number of iteration cycles, and (iii) the CC network with the corresponding teaching method can be trained to any desired accuracy at the expense of increasing the number of hidden nodes, sometimes to impractical values.

As a next step, 600 training patterns were used for the NSLS's X-ray storage ring, which has 48 orbit monitors and 40 correctors. In this case, the $\vec{X}$ inputs were measured for the 600 random $\vec{\Theta}$ outputs. Based on the previous results, a 3 layer shortcut connected network with 24 hidden nodes was used and Quickprop was chosen as a training method. After 2200 iterations cycles the Nnet was trained to $44\mu$ orbit accuracy.

## 7 REFERENCES


[1] D. Hammerstrom, "Neural networks at work", IEEE Spectrum, June 1993, p.26.

[2] P.K. Simpson, "Artificial Neural Systems", Pergamon Press, 1989.

[3] A. Zell, G. Mamier, M. Vogt, N. Mache, R. Hübner, K.U. Herrmann, T. Soyez, M. Schmalzl, T. Sommer, A. Hatzigeorgiu, S. Döring, D. Posselt, M. Reczko, M. Riedmiller, M. Seemann, M. Ritt, J. DeCoster, "SNNS User Manual V3.2", Univ. of Stuttgart, Dept. IPVR, 1994.

[4] Aharon Friedman and Eva Bozoki, "Use of Eigen Vectors in Understanding and Correcting Storage Ring Orbits", NIM A344, p.269, 1994.

[5] Eva Bozoki and Aharon Friedman, "Orbit Correction in Accelerators/Storage rings using Neural Networks:, AIP Conference Proceedings, V315, 1994.