*Published in the Proceedings of the*
*International ICSC/IFAC Symposium on*
*Neural Computation (NC'98)*
*Vienna, 1998*
*ISBN # 3-906454-15-0*

# An Addition to Backpropagation for Computing Functional Roots

***Lars Kindermann***

*FORWISS - Bavarian Research Center for Knowledge-Based Systems*
*Am Weichselgarten 7, 91058 Erlangen, Germany*

*email: lars.kindermann@forwiss.de*
*http://www.forwiss.de/~kinderma*

## Abstract

Many processes are composed of a n-fold repetition of some simpler process. If the whole process can be modeled with a neural network, we present a method to derive a model of the basic process, too, thus performing not only a system-identification but also a decomposition into basic blocks. Mathematically this is equivalent to the problem of computing iterative or functional roots: Given the equation *F(x)=f(f(x))* and an arbitrary function *F(x)* we seek a solution for *f(x)*. A special topology of multilayer perceptrons and a simple addition to the delta rule of backpropagation will allow most NN tools to compute good approximations. Applications range from data analysis within chaos theory to the optimization of industrial processes, where production lines like steel mills often consist of several identical machines in a row.

## 1 Functional Roots or Fractional Iterations

The concept of the well known square root of a real number can easily be extended to functions. This is an important part of the theory of *functional equations*.

### 1.1 Definition

Given an arbitrary function $F(x): \mathbb{R} \to \mathbb{R}$, the function $f(x)$ with

$$f(f(x)) \equiv F(x) \tag{1}$$

is called a *functional* or *iterative root* of $F$.

Also higher order roots can be defined. For

$$f^k(x) = f(f(\ldots f(x)\ldots)) \equiv F(x) \tag{2}$$

the function $f = F^{1/k}$ is a *k-th iterative root* of $F$.

Some simple examples:

$$\begin{aligned} F(x) &= x+1 \quad \Rightarrow \quad f(x) = x + 1/2 \\ F(x) &= x^2 \quad \Rightarrow \quad f(x) = x^{\sqrt{2}} \\ F(x) &= x^2 + 1 \quad \Rightarrow \quad f(x) = ? \end{aligned} \tag{3}$$

The last equation can probably not be solved analytically in a closed form. But it can be shown that iterative roots of all orders exist for at least all continuous and strictly increasing real valued functions [1].

However, some regularization may be necessary to yield unique solutions. In higher dimensions for $F: \mathbb{R}^n \to \mathbb{R}^n$ the existence of $F^{1/k}$ cannot be guaranteed. But there may be found "near fits", either

$$g(f(x)) \equiv F(x) \text{ with } \|f, g\| \to min \tag{4}$$

or

$$f(f(x)) \equiv G(x) \text{ with } \|F, G\| \to min \tag{5}$$

or a mix of both, depending on the desired application.

Finding functional roots can also be considered as the *inverse problem of iteration*: If $F$ is the *k-th iteration* of $f$, $f$ is the *k-th fractional iteration* of $F$.

### 1.2 Sample Applications

There are many problems, both from theory and practical applications, which can be related to solving functional roots. Here we present two examples to demonstrate the usefulness of this concept apart from pure mathematics.

#### 1.2.1 Chaos Theory

Iterated functions play a key role in chaos theory. The logistic equation

$$x_{n+1} = \lambda x_n (1 - x_n) \tag{6}$$

generates chaotic sequences $x_n$ for $\lambda > 3.57$ and the famous Mandelbrot-set results from the same iteration, only with complex-valued $\lambda$ [2].

Given numerical sequences $x_n$ of unknown origin, it is possible to reconstruct the iterative function $f: \mathbb{R} \to \mathbb{R}$ $x_n = f(x_{n-1})$ graphically or model it with a neural network. But if only partial sequences are available e.g. $n = 3, 6, 9\ldots$ only $F: \mathbb{R} \to \mathbb{R}$ $x_n = F(x_{n-3})$ can easily be reconstructed and it is necessary to compute the third iterative root of $F$, in order to get the fundamental generative function $f$ of this sequence.

### 1.2.2 Steel Processing

Industrial processes are often n-fold repetitions of some simple process, too. Figure 1 shows an example of a steel mill where stripes of metal are rolled in a sequence of up to seven identical stands from their initial thickness of some centimeters down to some millimeters.

Due to technical reasons it is impossible to measure some parameters like the profile of the stripes in between the stands - but that knowledge is necessary for optimal process control. Therefore a model of a single stand must be generated from the measured values of the incoming and outgoing material and the fact that the transformation occurred in a number of identical steps. The whole process line can successfully be modeled by a neural network [3], thus the function $F$ is well known but computation of internal values currently relies on mathematical models which lack the performance of an adaptive net.

A desired model-free method to simulate a single stand $f$ and retrieve the internal state $x_i$ of the steel needs to compute something like the n-th functional root of the whole array. This task is complicated by some additional parameters like the rolling pressure $p$ which is set to different values for each stand:

$$x_{out} = F(x_{in}, p_1 \ldots p_n) = f(\ldots f(f(x_{in}, p_1), p_2) \ldots, p_n) \qquad (7)$$

We call $f$ the *parametrized iterative root* of $F$.

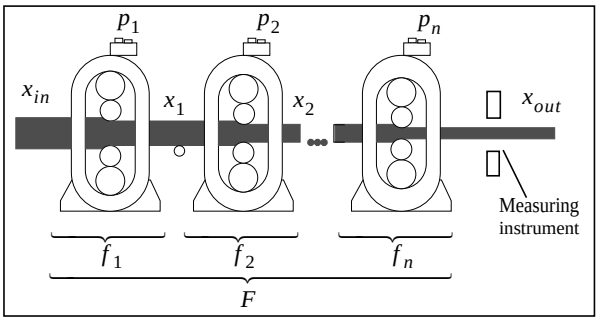


**Figure 1:** Model of a steel mill: A block of steel with known property $x_{in}$ is transformed by $n$ stands to a stripe with the measurable property $x_{out}$. Intermediate values $x_i$ are not accessible, but important to know for optimal process control. System identification with neural networks can easily model $F$ but revealing a description of a single stand $f$ is equivalent to computing parametrized iterative roots of $F$.

# 2 Designing a Backpropagation Network to solve Iterative Roots

Due to the fact that there are currently no standard numerical algorithms available to solve these problems, we have developed a method which uses MLPs to compute an approximation of iterative roots from given data.

## 2.1 Defining the Topology

Given a data set $(x, y)$ where $y = F(x)$, it is a standard problem for neural networks to find an approximation for $F(x)$. Usually a MLP with one hidden layer (1-N-1, N denotes the number of hidden units) is sufficient, if it has enough neurons.
Performing the same task with a 1-N-1-N-1 network as shown in Figure 2 will of course yield a similar result. Such a network could be interpreted as a chain of functions $y = F(x) = g(f(x))$, where the first 1-N-1 subnet represents $f$ and the second one $g$. To allow arbitrary outputs of the single layer neurons, their activation functions are set to identity. But of course there is yet no reason why $f$ and $g$ should have any other special relationship.

## 2.2 Coupling the Weights

If we want to have $f(x) \equiv g(x)$ then each layer has to act identically. To achieve this, the corresponding weights of the subnets have to be identical. Backpropagation can easily be modified in a way, that the same weights are shared between several connections ($w_i = w_j$), which is a common practice in time-delay networks for example [4]. But because of better training performance and to handle the cases described by equation (4), we force them to approach each other slowly instead, by adding an additional term to the standard delta-rule [5] for weight-update:

$$\delta w_i = \delta w_{i\ \text{backprop}} + \alpha(w_j - w_i) \qquad (8)$$

where $w_j$ is the current value of the corresponding weight within the other layer and $\alpha$ an user definable parameter, we call *coupling factor*.

This is equivalent to the addition of a penalty-term, the sum of the squared differences of corresponding weights, to the error function, but easier to implement and compute in this local manner.

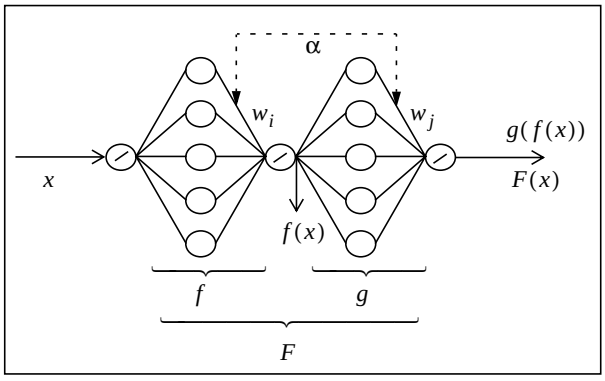


**Figure 2:** A MLP with a 1-N-1-N-1 topology splits the net function $F$ in two separable parts, $f$ and $g$. Coupling all corresponding weights forces $f$ and $g$ to become identical and thus a functional root of $F$.

## 2.3 Training the Net

We found it most successful to cut the training process in two steps. First, the network is trained with standard backpropagation without weight coupling ($\alpha = 0$) until it reproduces $F(x)$ with appropriate accuracy. Then the factor $\alpha$ is slowly increased while keeping the whole net approximating F continuously.

Setting $\alpha = 1$ immediately often results in a failure of the net to learn $F$, especially when working with higher order iterations, i.e. using networks with many single neuron layers. These bottlenecks strongly inhibit the error backpropagation and the fewer degrees of freedom implied by that "hard" weight coupling seem to disturb the learning ability additionally.

## 2.4 Retrieving the Results

Now only the first (1-N-1) subnet is needed to compute the desired functional root $f(x)$. It can be extracted from the whole net and be used to tabulate the function or serve as a subroutine in another application.

# 3 A simple Demonstration

Figure 3 shows how the network from Figure 2 is trained with a data set consisting of 11 evenly spaced x-values from the interval [0..1] and corresponding y-values computed with the function $y = F(x) = x^2 + 1$. Not surprisingly, this net with its 1-5-1-5-1 topology is able to approximate $F(x)$ on the whole interval very well.

But evaluating the transfer functions of the subnets reveals completely different $f$ and $g$. In fact, every new training with different weight initialization yields a completely altered internal behavior of the net. But after turning on weight-coupling, i.e. setting $\alpha$ to a small positive value, the corresponding weights of the subnets approach each other slowly until $f$ and $g$ become identical. And because the whole net $F$ still does a good approximation of the data, $f$ is now an estimate of the functional root of $x^2 + 1$.

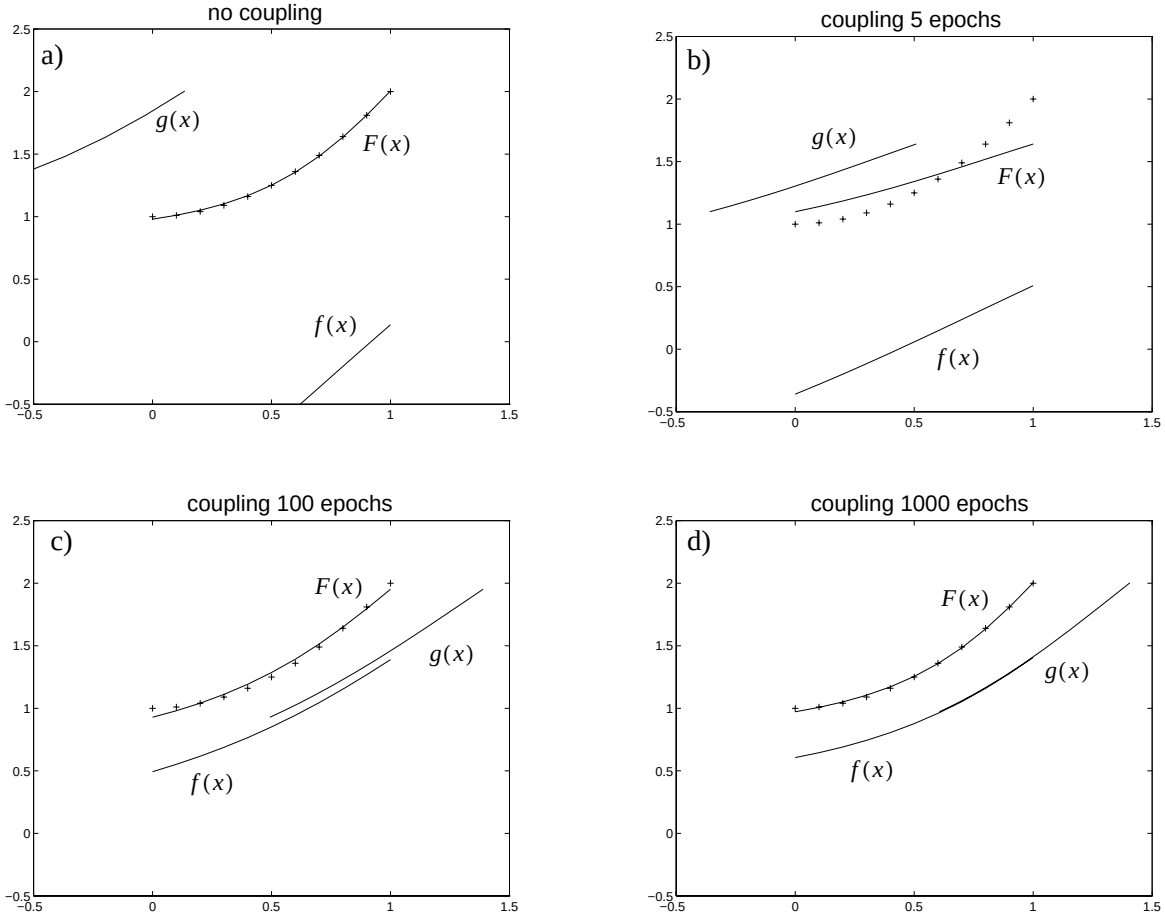


**Figure 3:** The effect of weight coupling. The trained MLP with $\alpha = 0$ from Figure 2 has learned the function $F(x) = x^2+1$ very well but the subnets *f* and *g* behave quite different (a). After turning on weight coupling, the accuracy of F first decreases (b), but *f* and *g* approach each other (c). Finally the approximation of *F* is satisfying again and *f* and *g* are identical now, thus representing the functional root of *F* (d).

# 4 Conclusions

We have presented a method to compute iterative roots with neural networks and demonstrated its ability to solve numerical examples. Higher order iterations, functions with local parameters and the performance of the method under noise are the next steps to further develop this field and evaluate its value for more complex applications. But even as simple applications as the shown above might be of broader interest as standard algorithms for computing functional roots are not easily available.

Due to their lack of explanation capabilities, neural networks are mostly used for black box simulations. In contrast, this application utilizes the network to gain *new knowledge* about a system, otherwise not accessible, by breaking the black box into smaller, easier parts which may help to understand the underlying principles which generated a specific data set.

A special advantage of the presented method is its simplicity: The additional term of the weight update rule in equation 8 can easily be implemented into most available NN tools to utilize them for computing functional roots.

This field is conceptually related to the simulation of dynamical systems, but addresses different questions. In conjunction with other methods it might help not only to reproduce a given dynamical system but to do so on a finer resolution. It also may help to decide, if a given time-series could result from a closed dynamical system, as the equations describing the temporal development of a system can be mapped to iterated functional equations which must possess iterative roots in order to comply with steadiness of physical processes.

Fractal compression methods also show some similarities to the concept of functional roots. They express complex data, images e.g. as the result of some iterative process. The compression task is to find a transformation which after multiple iterations yields the desired result and is described by as little information as possible.

There may also exist links to neuroscience. It is known that there are strong reciprocal links between different cortical areas and also bidirectional vertical connections between cortex and thalamus. Higher cognitive tasks often are not performed in a strictly feedforward manner but activation flows forward and back through the same network structures several times. It should be investigated if some of these tasks can be explained as iterative processes and learning in the brain thus involves finding “iterative roots”.

**Acknowledgements**

This research was sponsored by the German Federal Ministry of Education, Science, Research and Technology under grant number 01 IN 505 B.

**References**

[1] Kuczama M., Choczewski B., Ger R., *Iterative Functional Equations*. Cambridge University Press, Cambridge, 1990.

[2] Feigenbaum M. J., *Universal Behaviour in Nonlinear Systems.* Los Alamos Science, Los Alamos, 1978

[3] Martinetz T., Protzel P., Gramckow O., Sörgel G., *Neural network control for steel rolling mills*. In Kappen B., Giele S., *Neural Networks: Artificial intelligence and industrial application*. Springer, Berlin, 1995.

[4] Waibel A., *Modular construction of time-delay neural networks for speech recognition*. Neural Computation 1:39-46, 1989.

[5] Rumelhart D. E., McClelland J. L., *Parallel Distributed Processing*. MIT-Press, Cambridge, 1984.