

Computing Iterative Roots with Neural Networks

Lars Kindermann

Email: Lars.Kindermann@forwiss.de

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

ABSTRACT

Many real processes are composed of a n-fold repetition of some simpler process. If the whole process can be modelled 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 $f(x)$. Solving this functional equation in a closed form is an exceptionally hard problem and often impossible, even for simple functions. Furthermore there are no standard numerical methods available yet. But 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 even of higher order iterative roots.

Applications range from data analysis within chaos theory (many chaotic systems are derived from iterated functions) to the optimization of industrial processes, where production lines like steel mills often consist of several identical machines in a row.

KEYWORDS: Algorithms and Architectures, Chaos

1. ITERATIVE ROOTS OR FRACTIONAL ITERATIONS

1.1 Definition

Given an arbitrary function $F(x)$, the function $f(x)$ with $f(f(x)) \equiv F(x)$ is the *functional* or *iterative root* of F . For $f^k(x) = f(f(\dots f(x)\dots)) \equiv F(x)$ the function $f = F^{1/k}$ is called the *k-th iterative root* of F . If F is the *k-th iteration* of f , f is the *k-th fractional iteration* of F . Some examples:

$$F(x) = x + 1 \Rightarrow f(x) = x + 1/2$$

$$F(x) = x^2 \Rightarrow f(x) = x^{\sqrt{2}}$$

$$F(x) = x^2 + 1 \Rightarrow f(x) = ?$$

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. The question of uniqueness must be answered less satisfying, as there are generally many solutions [1]. Restrictions have to be added to yield unique solutions of iterative roots.

1.2 Applications

1.2.1 Chaos Theory

Iterated functions play a key role in chaos theory. The logistic equation $\mathbb{R} \rightarrow \mathbb{R} \quad x_{n+1} = \lambda x_n(1-x_n)$ generates chaotic sequences x_n for $\lambda > 3.57$ and the famous Mandelbrot-set results from the same iteration, only with complex-valued λ [2].

Given numerical sequences x_n of unknown origin, it is possible to reconstruct the iterative function $f: \mathbb{R} \rightarrow \mathbb{R} \quad x_n = f(x_{n-1})$ graphically or model it with a neural network. But if only partial sequences are available e.g. $n = 3, 6, 9 \dots$ only $F: \mathbb{R} \rightarrow \mathbb{R} \quad x_n = F(x_{n-3})$ 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 centimetres down to some millimetres.

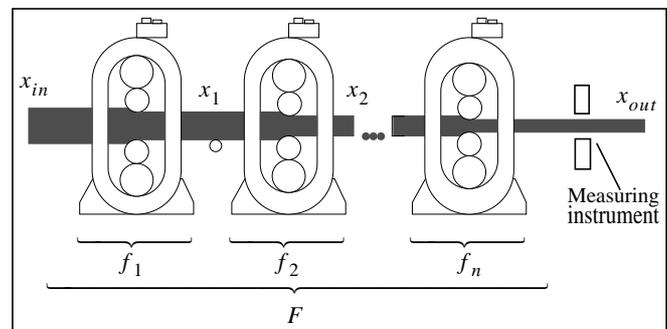


Figure 1: Model of a steel mill: A block of steel with known property x_{in} is transformed by n stands to a stripe with the measurable property x_{out} . Intermediate values x_n are not accessible, but important to know for optimal process control. System identification with neural networks can easily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

Due to technical reasons it is impossible to measure parameters like the profile of the stripes in between the stands - but the knowledge of them 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 knowledge that the transformation occurred in a number of identical steps. The whole process line can successfully be

modelled 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.

2. DESIGNING MLPs TO SOLVE ITERATIVE ROOTS

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) is sufficient if it has enough neurons.

It is clear that performing the same task with a 1-N-1-N-1 network as shown in Figure 2 should 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. Of course there is no reason why f and g should have any other special relationship.

2.2 Coupling the weights

If we manage to make $f(x) \equiv g(x)$ then each layer would act identical, and perform like the functional root of the whole net. To achieve this, the corresponding weights of the subnets have

to be identical. This can be forced by adding an additional term to the delta-rule [4] of the weight-update within the backpropagation algorithm:

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

where w_j is the current value of the corresponding weight within the other layer and α a user definable parameter, we call *coupling factor*.

This is equivalent to adding an additional penalty-term, the sum of the squared differences of corresponding weights, to the error function used for the backpropagation algorithm but easier to implement and compute in this local manner.

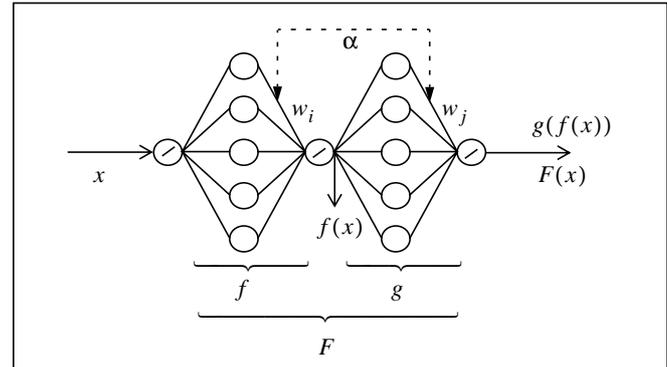


Figure 2: A MLP with a 1-N-1-N-1 topology splits the net function F in two separable parts, f and g . Coupling all corresponding weights forces f and g to become identical and thus a functional root of F .

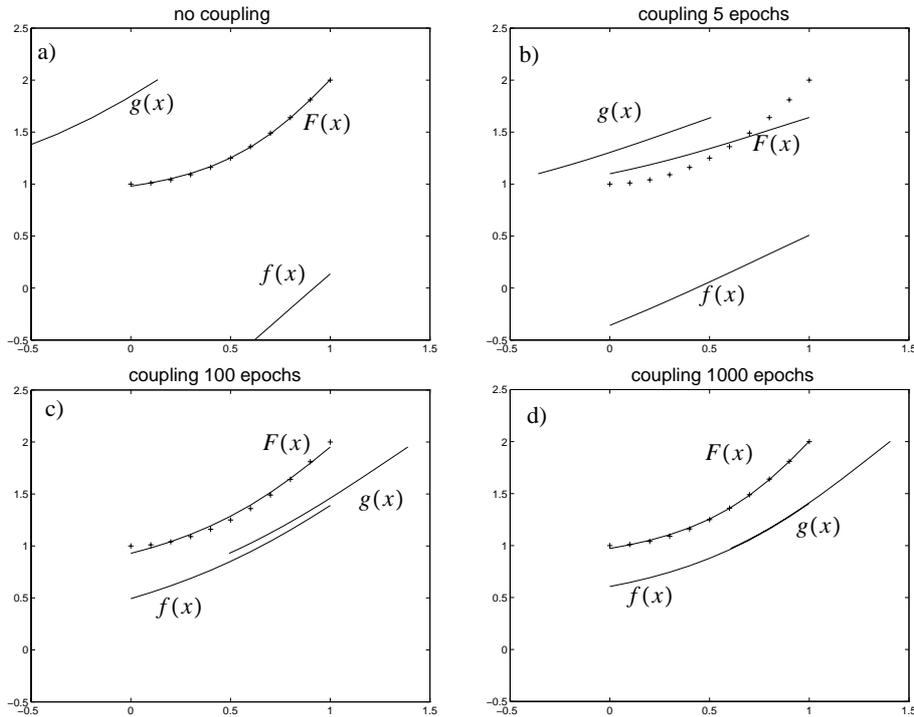


Figure 3: The effect of weight coupling. The trained MLP with $\alpha = 0$ from Figure 2 has learned the function $F(x) = x^2 + 1$ very well but the subnets f and g behave quite different (a). After turning on weight coupling, the ac-

curacy 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).

3. AN EXAMPLE

Figure 3 shows how the network from Figure 2 is trained with a data set consisting of 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 behaviour of the net. But after turning on weight-coupling, i.e. setting α 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 $x^2 + 1$.

Trying to solve this problem with $\alpha = 1$ which means forcing corresponding weights to be always identical, results in a difficult learning task: The bottlenecks of the single neuron layers make backpropagation perform very badly and often such a network completely fails to approximate $F(x)$. The performance is best when starting the training with $\alpha = 0$ until the net has learned $F(x)$ and then increasing its value slowly towards one.

4. CONCLUSIONS

We have presented a method to compute iterative roots with neural networks and showed its ability to solve simple numer-

ical 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.

There are other neural methods, recurrent networks for example, which should yield similar functionality to the algorithm presented above. A main reason why we went this way first, is the possibility to implement it easily into most available tools: After adding the weight coupling ability to any MLP simulator, it is possible to utilize it for computing functional roots.

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 by breaking the black box into smaller, easier parts which may help to understand the underlying principles which generated a specific data set.

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] Martinez 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] Rumelhart D. E., McClelland J. L., *Parallel Distributed Processing*, MIT-Press, Cambridge, (1984)