In: Neural Information Processing, ICONIP2001 Proceedings, Fudan University Press, Shanghai 2001, Vol 2, pp. 1075-1078.

A framework for solving functional equations with neural networks

Lars Kindermann[1]
kindermann@reglos.de
www.reglos.de/kindermann

Achim Lewandowski[2]
lewandowski@alewand.de
www.alewand.de

Peter Protzel[2]
peter.protzel@e-technik.tu-chemnitz.de
www.infotech.tu-chemnitz.de/~proaut

[1]RIKEN Brain Science Institute, Lab for Mathematical Neuroscience
Wako-Shi, Saitama 351-0198, Japan

[2]Dept. of Electrical Engineering and Information Technology
Chemnitz University of Technology, 09107 Chemnitz, Germany

Abstract

In his „essay towards a calculus of functions" from 1815 Charles Babbage introduced a branch of mathematics now known as the theory of functional equations [1]. But since then finding specific solutions for a given functional equation remained a hard task in many cases. For one of his examples, the now famous „Babbage equation" φ(φ(x))=x, which solutions φ are called „the roots of identity" and the more general equation φ(φ(x))=f(x) which defines kind of a „square root" of some given function f, we have previously shown that this type of equation can be solved approximately by neural networks with a special topology and learning rule. Here we extend that method towards a wider range of functional equations which can be mapped in similar ways to neural networks too. The method is demonstrated on - but not limited to - multilayer perceptrons. We present a first sketch of this ideas here on some important equations.

1 Introduction

Functional equations are a relatively unpopular area of mathematics. This is not due to a lack of importance: Extending linear algebra which deals with linear functions, functional algebra covers a much more general domain.

Usually dynamical systems are described by differential equations in a continuous time domain. For discrete time systems on the other hand, the dynamics is defined by a difference equation or an iterated map. Constructing a trajectory or determining other properties of the system requires dealing with functional equations.

Contrary to differential calculus or linear algebra, functional equations are rarely employed to solve practical problems. This may be due to the technical difficulties of the functional calculus. Functional equations are often *very* hard to handle. For many of them mathematics has not yet determined a general solution, even the questions of existence and uniqueness are not solved.

One of the simplest functional equations possible is $\varphi(\varphi(x)) = f(x)$, which expresses the direct extension of the concept of *square roots of a number* towards the calculus of functions: The function φ is something like a *square root of the function* f, also often called iterative root or functional root of f.

A 2001 survey paper on functional equations states: *„...one should not expect results on iterative roots in a general situation. In fact, even roots of polynomials are not described. Even worse: we do not know whether every complex cubic polynomial has a square root..."* [3]

Facing an engineering problem which required solutions of this type of equation, we have successfully developed methods to solve this equation at least numerically with the aid of neural networks. [4]

That work inspired us to compile more neural network solutions for similar mathematical problems, providing a common framework for mapping functional equations to neural networks.

2 Neural Networks as Building Blocks

Here we present some examples of functional equations and a network topology which will „solve" them.

We will use this terminology: x expresses a real number or some n-dimensional vector of real numbers. $f(x)$ is a ***known*** function, $\varphi(x)$ denotes an unknown function, the desired solution of the equation in question. c is a given constant. In the figures circles will represent nonlinear, usually sigmoid neurons while squares symbolize linear transfer functions.

Apart from the case where explicit functions are given as a formula, practical applications often relay on measurements or sampled data. If the known function f is given only implicitly as input - output data pairs defined by a table of $(x, f(x))$ values, the *training set* in neural network terminology, solving a functional equation containing f becomes part of a regression problem: There may be noise or errors in this data and one has to deal with problems like generalization and overfitting. Neural networks have proven to be highly successful in this context and lots of methods are available, which then naturally combine with the functional equation capability described here to provide practical solutions for specific applications.

2.1 The Inverse of a Function

$$\varphi(f(x)) = x$$

This equation is solved by the inverse of the given function: $\varphi = f^{-1}$. Finding an inverse with neural networks is a well known procedure. The most simple technique is just exchanging inputs and targets of the training set. But also several analytic methods have been described to invert a ready trained net by calculating the weights directly. [5] This provides a basic procedure for several of the following examples.

2.2 Schröder's Equation

$$\varphi(f(x)) = c\varphi(x)$$

The Schröder equation [6] represents the eigenvalue problem of functional calculus. It is one of the most important functional equations and useful for linearization of various other functional equations. [2]

We suggest two methods for addressing this with neural networks. The first can be used when it is known that the desired solution φ is a bijective mapping and possesses an inverse: Then the equation can be written as

$$f(x) = \varphi^{-1}(c\varphi(x))$$

and be mapped to this neural network structure.

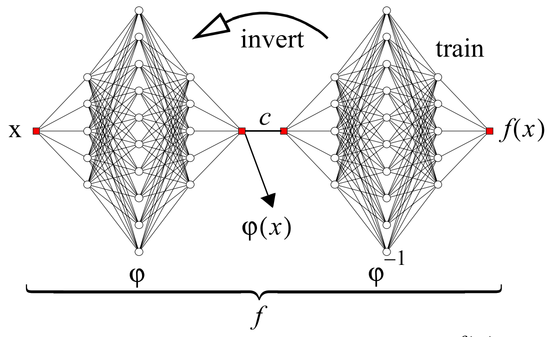

Figure 1: This network as a whole represents $f(x)$, but only the second part is trained by backpropagation. The constant c is mapped to a weight which is kept constant. During the training the right subnet is concurrently inverted and the result is transferred to the left part which finally - if the process converges - approximates the sought solution φ.

The other method we propose here represents each side of an equation as a separate networks, which are trained alternately towards the output of each other and corresponding parts are linked by *weight sharing*. The part which represents $f(x)$ has to be trained in advance. The principle is the same as described in section 2.7.

2.3 Abel's Equation

$$\varphi(f(x)) = \varphi(x) + c$$

The Abel equation is another very basic equation of functional calculus. It also represents some way to linearize other more complicated problems [2]. This equation can be handled similar to the previous one just by mapping c to a fixed bias instead of the weight between the two subnets which is kept fix at the value 1 now.

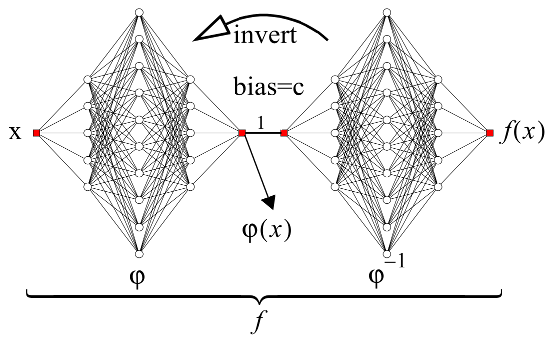

Figure 2: For the Abel equation the constant is represented as a fixed bias.

2.4 Iterative Roots

$$\varphi(\varphi(x)) = f(x)$$

Iterative roots belong to the group of iterative functional equations. They represent the inverse problem of iteration and have applications in dynamical systems, chaos and modelling of industrial processes.

They can be easily mapped to a multilayer network which consists of two identical subnets in a row. The only problem is to train the whole net to approximate $f(x)$ and keep all the weights in both subnets identical. This can be achieved by different training methods based on backpropagation e.g. weight sharing or adding penalty terms for regularization [7].

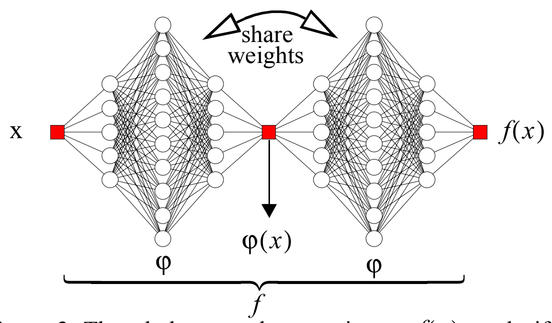

Figure 3: The whole network approximates $f(x)$, and - if kept identical by weight sharing - each subnet approximates the iterative root of f.

2.5 Babbage Equation

$$\varphi(\varphi(x)) = x$$

This special case of 2.4 with $f(x) = x$ is the famous Babbage equation. Charles Babbage was a pioneer in the area of functional equations. In his „Essay towards the calculus of functions“ from 1815 [1] he presented several functional equations together with some solutions. The (many) solutions φ are called the „roots of identity“, corresponding to the „roots of unity“ in the complex number domain.

2.6 Fractional iteration

More general, the fractional iterate $f^{m/n}$ of a function is defined by the equation

$$\varphi^n(x) = f^m(x)$$

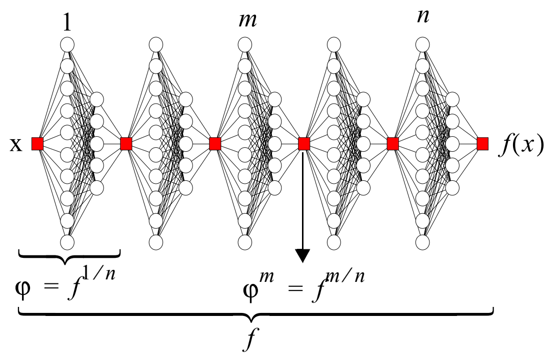

Figure 4: This **multi**layer network computes the fractional iterates of its target function.

Training methods based on backpropagation with weight sharing or penalty terms for regularization and direct 2nd order gradient descend have been developed by us which can compute up to the 10th root of a given function. [7]

2.7 Commuting Functions

$$\varphi(f(x)) = f(\varphi(x))$$

This equation expresses the fact that the order of two functions may be interchanged without changing the output.

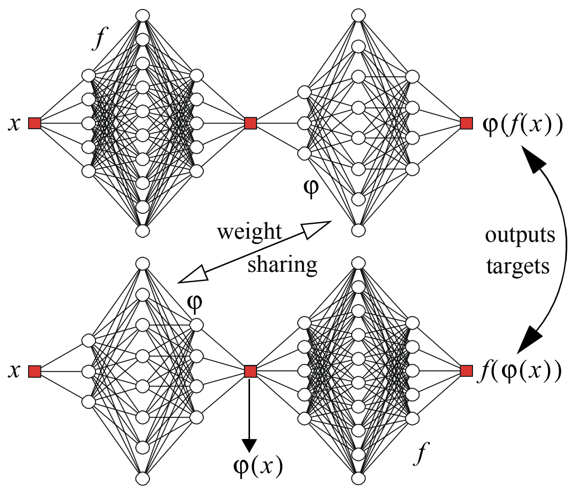

Figure 5: The equation is expressed as two networks which are trained to deliver identical results by using the output of each as the target of the other. The f networks are pre-trained to the function $f(x)$ and then kept fix, the φ networks are trainable but forced to stay equal by some weight sharing mechanism.

This shows that there may be many solutions to a given functional equation, sometimes even depending on an arbitrary function. So this network alone will find just one arbitrary solution. However, combined with other knowledge, perhaps some another functional equation, a unique solution may be constructed.

2.8 Periodic Equation

$$\varphi(x) = \varphi(x+c)$$

This equation is solved by every periodic function with period c. Thus it is not very interesting in itself, but in a context with other equations it provides a method for defining periodic behaviour. Notice that there is no known function present in this equation.

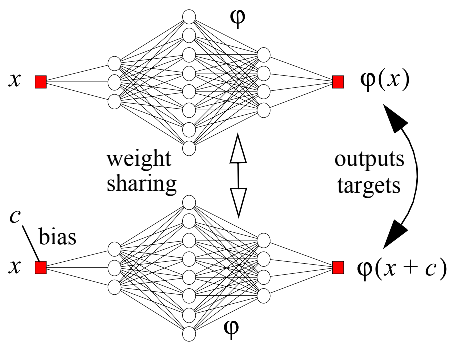

Figure 6: These coupled networks should converge to some periodic functions when „trained". The bias c determines the period.

2.9 Gamma like functions

$$x\varphi(x) = \varphi(x+1)$$

For integer x this equation defines $x!$, for real x a known solution is Euler's Gamma function $\Gamma(x)$, but there are other solutions possible. The network structure is only slightly different from above.

2.10 Feigenbaums Equation

$$c\varphi(x) = \varphi(\varphi(cx))$$

This equation plays an important rule in the theory of quadratic mappings and renormalization theory of dynamical systems [8].

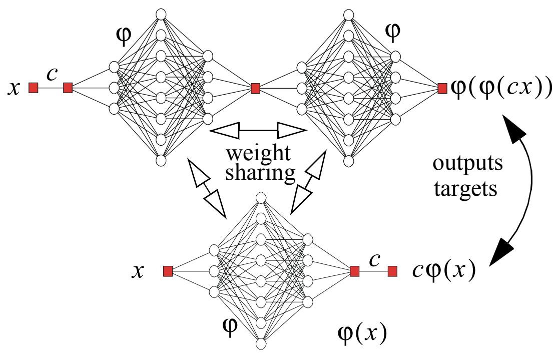

Figure 7: A network for finding solutions of the Feigenbaum equation.

2.11 Systems of Equations

From the examples above one should get an idea how to construct neural networks which correspond to a functional equation. Sometimes several functional equations are used to describe a problem. If they all contain the sought function $\varphi(x)$, then mapping them to different networks and training them all together (alternatively) while matching the φ subnets by weight sharing can find a solution to this combined problem.

2.12 Existence and Uniqueness

Not every functional equation posesses a solution. And if it has, it may not be unique. Many solutions depend on an arbitrary function. [2] If something about the desired solution is known there is a trick to increase the probability that the network will come up with this one: Instead of initializing the weights with random values, pretraining with a function similar to the expected one can help to drive the learning process towards that direction. [7]

3 Conclusions

This paper shows how to address functional equations with neural methods, some of them have been proven to deliver successful results, other have yet to be verified. Some future work will be devoted to them.

Most interesting is that these questions go beyond the traditional use of neural networks in function approximation and classification, where the questions asked to the net are from the same domain as the available training data. They perform merely interpolation and extrapolation. If one had more or better data, the problem would diminish. These applications presented here on the other hand use neural networks to answer questions of a completely different kind and on a higer level of abstractness. They can be regarded as a kind of *knowledge extraction* from the pure data where the solutions φ may give insight in otherwise hidden processes underlying the observable data. This holds definitely true in dynamical systems, which are a mayor driving force for the renaissance of functional equations within the last few years.

This neural network framework presented here may provide valuable tools for the application of functional equations in science and technology as other numerical methods are rare. A method based on solving fractional iterations for modelling metal strip profiles in steel rolling mills has recently been patented by a mayor company and is used in industrial production now. [4]

4 References

An extensive compilation of references mainly on iterative roots but also other functional equations together with many links for direct download of papers or abstracts can be found at our website: *http://www.reglos.com/kindermann/ffx.html*

[1] C. Babbage, *An Essay towards the Calculus of Functions.* Philosophical transaction of the Royal Society London 105: 389-424, 1815

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

[3] K. Baron, W. Jarczyk, *Recent results on functional equations in a single variable, perspectives and open problems*. Aequationes Math. 61: 1-48, 2001

[4] L. Kindermann, *Computing Iterative Roots with Neural Networks.* Proceedings of the Fifth Intl. Conf. on Neural Information Processing, ICONIP'98 Vol. 2: 713-715, 1998

[5] J. Kindermann & A. Linden, *Inversion of neural networks by gradient descent.* Parallel Computing, 14(3): 277-286, 1990

[6] E. Schröder, *Über Iterierte Funktionen*. Math. Ann. 3: 296-322, 1871

[7] L. Kindermann, A. Lewandowski, P. Protzel, *A Comparison of Different Neural Methods for Solving Iterative Roots.* Proc. Seventh Int'l Conf. on Neural Information Processing, ICONIP'2000, Vol2: 565-569, 2000

[8] K.M. Brigs, T.W. Dixon, G. Szekeres, *Analytic solutions of the Cvitanovic-Feigenbaum and Feigenbaum-Kadanoff-Shenker equations.* International Journal of Bifurcation and Chaos, Vol. 8, No. 2: 347-357, 1998