Computing Iterative Roots with Second Order Training Methods

Lars Kindermann

reglos.com
kindermann@reglos.com
www.reglos.com/kindermann

Peter Protzel

Dept. of Electrical Engineering and Information Technology
Chemnitz University of Technology, Germany
peter.protzel@e-technik.tu-chemnitz.de
www.infotech.tu-chemnitz.de/~proaut

Abstract

Iterative roots are a valuable tool for modeling and analyzing dynamical systems. They provide a natural way to construct a continuous time model from discrete time data. However, they are in most cases extremely difficult to compute analytically. Previously we have demonstrated how to use neural networks to calculate the iterative roots and fractional iterations of functions. We used a special topology of MLP's together with weight sharing. This paper shows how adding a regularization term to the error function can direct any backpropagation based training method to the same result but in a fraction of epochs when using advanced 2-nd order learning rules.

1 Iteration = Discrete Time Evolution

Among lots of other applications iterative roots of functions are the key for embedding time discrete systems into continuous time or changing the time base.

Suppose you have collected market data on a monthly base and have created a pretty good neural model for predicting next month sales of your company. But due to an increased business speed you are asked to make predictions now on a weekly base. Will you have to wait until you have sampled enough data again each week for training a weekly model? Or will you just interpolate the weeks from a month? But for nonlinear dynamics this is pretty difficult without knowing the equations of the system. But it is possible to compute this in a model free way indeed!

The dynamics of discrete time systems is best described in terms of iteration. The time evolution of a system is described by a function

$$x_{t+1} = f(x_t)$$

x are vectors in the state-space of the system. This may also take the form of a difference equation $x_{t+1} = x_t + \Delta(x_t)$, but it can easily converted to the form above. Computing the trajectory of the system from a starting point x_0 means just applying f over and over again, f is the *time-one mapping* function of the system. Using a well known notation, $x_2 = f(f(x_0))$ can be written as $x_2 = f^2(x_0)$. Then the state of the system at time t is given by

$$x_t = f^t(x_0)$$

f^t is called the t-th iteration of f. The common notation for the inverse of a function f^{-1} fits nicely into this picture too: Computing the previous state of the system at time $t = -1$ means applying the inverse of f to x_0:

$$x_{-1} = f^{-1}(x_0) \qquad (1)$$

Let $f^0(x) = x$ denote the identity, which also fits nicely into this picture.

Thus f^t is the time evolution operator for all integer times t.

Remark: From chaos theory we know that iterating even pretty simple functions can exploit extremely complex behavior, like the logistic equation $x_{t+1} = ax_t(1-x_t)$ which is the schoolbook example for chaos.

2 Iterative Roots

Iterative roots are the extension of the roots of numbers to the domain of function spaces and they are defined in terms of a *functional equation*:

Definition: Given an arbitrary mapping f of a set S to itself, a solution φ of the equation

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

$x \in S$, is called a *functional* or *iterative root* of f.

In general, for the equation

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

the solution φ is called a *n-th iterative root* of f and the formal notation $\varphi = f^{1/n}$ is used. Similar a solution of the equation

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

can be written as $\phi(x) = f^{m/n}(x)$, a *fractional iteration* of f.

This extends the common notation of iterates or „powers“ of functions from the well known integer exponents to arbitrary fractional numbers.

The mathematics of iterative roots is rather difficult. They appeared first 1815 in the Babbage equation

$$\varphi^2(x) = x$$

which solutions are called „the roots of identity“. This simple case of an iterative root already demonstrates that there is no uniqueness attached to a solution: In addition to the obvious solution $\varphi(x) = x$ there are infinitely many other solutions, e.g. $\varphi(x) = a - x$. Very often solutions depend on some arbitrary function.

On the other hand the question for the existence of iterative roots of a given function f turns out to give the surprising answer, that „almost no“ function posesses iterative roots, (mathematically spoken): the subset of iterates is nowhere dense in most function spaces, but proofs are often difficult.

Something is known for real valued functions, e.g continuous and monotonically rising functions have continuous and monotonic roots of all orders. In general, each case has to be examined carefully.

A 2001 survey article on the current state of the research on iterative roots 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 From Discrete to Continuous Time

In chapter 1 we described the temporal evolution of a discrete time dynamical system by terms of iterating a self mapping function. Sometimes the question arises if such a discrete time dynamical system can be embedded into continuous time. In fact this should be possible for all physical systems as physical time is considered to be a continuous flow. (Only philosophers and quantum mechanics may be concerned). But in every practical case we realize every system only in finite time steps because of a technically limited sampling rate for observations. However, the final desired description is a continuous trajectory $x(t, x_0)$, or at least some differential equation describing the dynamics, which can be integrated to yield particular solutions.

The set of all f^n with $n \in N$ forms the iterative semigroup of f and can be embedded into what is called the*continuous iterative semigroup* of f with the notation $^t(x)$ with real t if these two conditions apply:

$$F^t(x) = f^t(x) \text{ for all } x \text{ and integers } t \in Z$$

and F conforms to the translation equation

$$F^{t_1}(F^{t_2}(x)) = F^{t_1 + t_2}(x)$$

As it is easily seen, the fractional iterates of f are behaving exactly this way and thus are a possible way of introducing continuous time to iterated maps.

Back to our problem from the beginning, if our model for the sales of next month is given by some function, we have to take the 4-th root of this model to get the one week model.

4 Computing Iterative Roots with Neural Networks

Neural networks are able to approximate any given (non-pathologic) function from example data. They are often called universal approximators. If the network from figure 1 is trained as a whole to approximate the function $f(x)$ and at the same time all the weights from the both subnets are kept equal then each subnet represents exactly the iterative root of f. Using backpropagation as the basic learning method two additional ways to enforce the regularization task were successfully demonstrated so far: Training only the second subnetwork and continuously copying the weights to the first net. If this process converges, both subnets are identical and represent the root of f. An alternate method implements a weight sharing mechanism between the corresponding weights of the two subnets which are otherwise treated like a normal MLP. This method converges at the same time to the target function and equal subnets. A mayor drawback of both methods is that higher order learning rules which are proven to give much better results than vanilla backpropagation, cannot be used because of the additional weight changes.

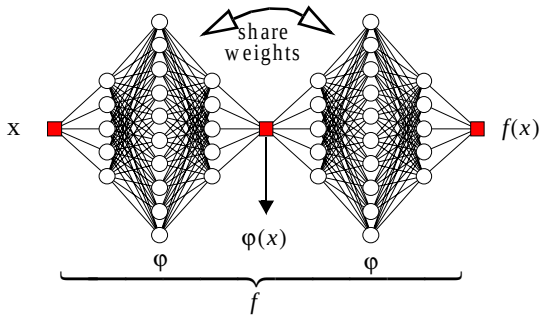

Figure 1: A MLP computes the iterative roots of f

The same arguments can be used for n-th iterative roots, where a composition of n networks is trained towards f. Taking then a row of only m of these sub-networks, this is equivalent to the fractional iteration $f^{m/n}$.

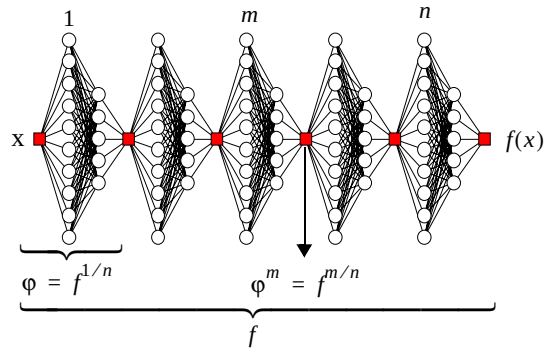

Figure 2: Computing a fractional iterate $\phi = f^{m/n}$

5 Regularization by Introducing an Error Term

When the regularization task of making both subnetworks as similar as possible, is formulated as an error term E_{reg} which adds to the usual approximation error E_{app} on the training data, most backpropagation based gradient descend algorithms can successfully handle this problem without further modifications.

$$E = E_{app} + E_{reg}$$

Let $w_{i,j}$ denote the i-th weight in the j-th subnet. Goal is to make all corresponding weights as equal as possible $w_{i,j} - w_{i,k} \to 0$ for all j, k. The index i has to include *all* weights of a subnet, including biases.

An appropriate error measure is the sum squared error of all corresponding weights.

m =Number of weights per subnet, n =number of subnets.

$$E_{regsum} = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} (w_{i,j} - w_{i,k})^2$$

There are $n(n-1)/2$ pairs of corresponding subnets, the mean squared error of two corresponding weights is

$$E_{regmean} = \frac{E_{regsum}}{m\frac{n(n-1)}{2}}$$

Another way is to use the sum of the variances of corresponding weights across all subnets. Starting with the mean value of corresponding weights

$$\overline{w_i} = \frac{1}{n} \sum_{j=1}^{n} w_{i,j}$$

the emperical variance is given by the following term:

$$s_i^2 = \frac{1}{n} \sum_{j=1}^{n} (w_{i,j} - \overline{w_i})^2$$

To normalize over all weights divide by the number of weights:

$$E_{regvar} = \frac{1}{m} \sum_{i=1}^{m} s_i^2$$

To construct the total error function a parameter $0 < \alpha < 1$ may be introduced which gives a choice between

$$E = \alpha E_{app} + (1-\alpha) E_{reg}$$

To force the network for a minimal approximation error of f set $\alpha \to 1$, if mainly equal subnets are desired a small α is appropriate. This choice is useful if there exist no exact iterative roots of f.

The gradient necessary for backpropagation is then given by

$$\frac{\partial E}{\partial w_{i,j}} = \alpha \frac{\partial E_{app}}{\partial w_{i,j}} + (1-\alpha) \frac{\partial E_{reg...}}{\partial w_{i,j}}$$

with e.g.

$$\frac{\partial E_{regmean}}{\partial w_{i,j}} = \frac{4}{mn(n-1)} \sum_{k=1}^{n} (w_{i,j} - w_{i,k})$$

Nothing else has to be changed in the backpropagation algorithm and derived methods, like quasi Newton gradient descend.

6 Example

The method is demonstrated on the fractional iterations of $f(x) = x^2$, because they can also be calculated analytically to compare the results. The inverse is calculated most easily with the same network just by exchanging inputs and outputs in the training data. The MLP consists of 8 subnetworks with an (1-8-1) structure, linear input and output neurons and sigmoid hidden units. Training method is quasi newton backpropagation. This results in an approximation error of 10^{-6} for $f(x)$. Training data consists of 100 x, y pairs from the in-

terval $[0, 1]$. The network usually converges to the solution within 400 training epochs.

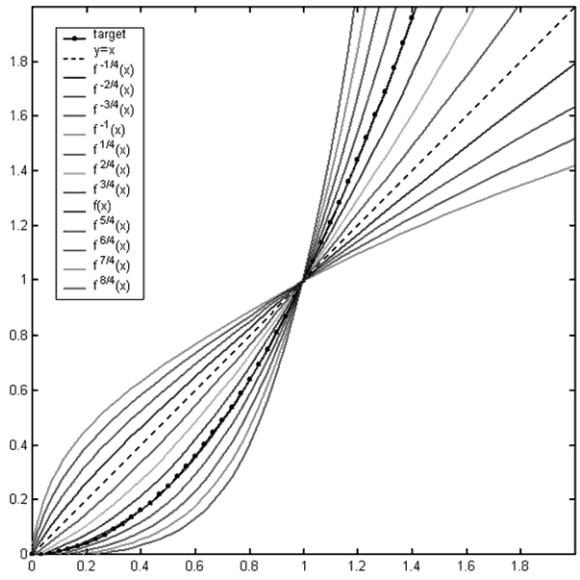

Figure 3: The fractional iterates of $f(x) = x^2$

7 Conclusions

The new method for computing iterative roots mean a significant advantage for applications over the weight sharing method. The table demonstrates the gain of performance compared to the older procedures. And because of the difficult analytic treatment and a lack of other available numerical methods so far, this should be valuable for all who are facing the problem of calculating iterative roots of functions.

References

[1] C. Babbage, *Essay towards the Calculus of functions.* Phil. trans. Royal Soc. London 105 (1815), 389-424

[2] Kuczama M., Choczewski B., Ger R., *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. (2001), 1-48

[4] G. Targonski, *An Iteration theoretical approach to the concept of time.* Colloques Internationaux du C.N.R.S. 229, Transformations ponctuelles et leurs applications, Toulouse (1973), 259-271

[5] G. Targonski, *Topics in iteration theory.* Vandenhoeck und Ruprecht, Göttingen 1981

[6] G. Targonski, *Progress of iteration theory since 1981.* Aequationes Math. 50 (1995), 50-72

[7] H. Kneser, *Reelle analytische Lösungen der Gleichung* $\varphi(\varphi(x)) = e^x$ *und verwandter Funktionalgleichungen.* J. reine angew. Math. 187 (1950), 56-67

[8] R.E. Rice, B. Schweizer & A. Sklar, *When is* $f(f(z)) = az^2+bz+c$ *for all complex* z*?* Amer. Math. Monthly 87 (1980), 252-263

[9] M.C. Zdun, *Continous iteration semigroups.* Boll. Un. Mat. Ital. 14 A (1977), 65-70

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