Finding iterative roots with a spiking neural network

Nicolangelo Iannella a,*, Lars Kindermann b,1

a Laboratory for Visual Neurocomputing, Brain Science Institute, RIKEN, 2-1 Hirosawa Wako-shi, Saitama 351-0198, Japan
b Laboratory for Mathematical Neuroscience, Brain Science Institute, RIKEN, 2-1 Hirosawa Wako-shi, Saitama 351-0198, Japan

Received 1 December 2004; received in revised form 25 March 2005
Available online 23 June 2005

Abstract

In recent years, both multilayer perceptrons and networks of spiking neurons have been used in applications ranging from detailed models of specific cortical areas to image processing. A more challenging application is to find solutions to functional equations in order to gain insights to underlying phenomena. Finding the roots of real valued monotonically increasing function mappings is the solution to a particular class of functional equation. Furthermore, spiking neural network approaches in solving problems described by functional equations, may be an useful tool to provide important insights to how different regions of the brain may co-ordinate signaling within and between modalities, thus providing a possible basis to construct a theory of brain function. In this letter, we present for the first time a spiking neural network architecture based on integrate-and-fire units and delays, that is capable of calculating the functional or iterative root of nonlinear functions, by solving a particular class of functional equation.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Spiking neural network; Iterative root; Functional equations; Distributed computing

1. Introduction

Typical industrial or manufacturing processes are composed of \( n \)-fold repetitions of simpler processes. The optimal control of certain manufacturing processes

\[
\phi(\ldots\phi(x)\ldots) = F(x),
\]

is often technically difficult, since this requires knowledge of the output at any intermediate stage. In most applications, the intermediate stage is usually inaccessible, as is the case for steel production. For these types of processes, a neural network can be used for both system identification and decomposition, in order to describe the entire process as a composition of the simpler building blocks. This is expressed as

\[
\psi(\ldots \psi(x) \ldots) = F(x),
\]

where \( x \) is the input and \( \psi \) represents the underlying building block of the entire process. Thus, the net-
work needs to not only learn the correct input–output relationship but also to seek a solution for \( \varphi(x) \). Mathematically, this is equivalent to the problem of calculating the \( k \)th iterative or functional root. Formally the iterative root, considered as the extension of the roots of number to the domain of function spaces, is defined as follows:

**Definition 1.** Given an arbitrary self mapping of a set \( X F : X \rightarrow X \), the solution to the functional equation \( \varphi(\varphi(x)) = F(x) \), \( x \in X \), \( \varphi \) is called the iterative root of \( F \). In general, the \( k \)th iterative root, denoted by \( \varphi = F^{1/k} \), is the solution to \( \varphi^k(x) = \varphi(...\varphi(x)... = F(x) \).

Calculating the solution to functional equations of the form (1) is not a trivial task, in fact they can be technically very challenging, since there is yet no standard analytical or numerical technique. This is further complicated by the fact that not all functions possess an iterative root [1]. What is currently known is that the iterative roots of any real continuous monotonically increasing function exists to all orders [2]. Fortunately, multilayer perceptrons (MLPs) have proven useful in their capacity to calculate iterative roots of (1) [3,4]. This raises both a challenging and interesting question: Can such a difficult computation be achieved by a network of spiking neurons?

In this letter, we will demonstrate that a simple network of spiking neurons, based upon a simple extension to a previously published architecture [5] can find the iterative root of (monotonically increasing) nonlinear functions; higher order and fractional roots will not be presented here. In Section 2, the network architecture is described. In Section 3, two simple learning algorithms and some results are discussed and presented. In Section 4 conclusions and future directions of focus are pointed out.

2. An integrate-and-fire network architecture

In this section, we will describe the network architecture used to solve the functional equation

\[
\varphi(\varphi(x)) = F(x),
\]

for \( \varphi \) and hence find the iterative root in a piecewise fashion. The spiking neural network (SNN) that we will soon describe is based upon an extension of a previously published network architecture [5]. This architecture spatially and temporally separates incoming spikes into different regions of the network, that in turn generate corresponding new spike pairs with any desired Inter Spike Interval (ISI). For this current case, the network consists of an input unit, a hidden unit, an output unit and two layers of spiking neurons denoted by layers \( A \) and \( B \) as depicted in Fig. 1. The input, hidden and output units simply transmit any received pair of spikes, they do not generate new spike pairs. Both the input and hidden units send pairs of spikes to all units of the subsequent layer (i.e., layer \( A \) receives from the input unit, then the output from layer \( A \) is sent to the hidden unit which in turn sends it to all units in layer \( B \)). The output from the hidden unit represents the iterative root. Model spiking neurons, represented as diamonds in Fig. 1, are the same as those used [5].

![Fig. 1. The spiking neural network architecture used to find the iterative root. The input, hidden and output units simply transmit spikes. Model spiking neurons are denoted by diamonds.](image-url)
In brief, the spiking neuron model, which is presented in Fig. 2, is composed of three parts; two input lines one of which has a delay of \( \tau_d \), an activation unit \( v_j(t) \) capable of firing a spike and two output lines, one possessing a delay \( \phi_j \). These three components are highly simplified representations of the dendrite, soma and axonal tree of biological neurons. The basic operation of the unit is to map an input spike pair \( r(t) = (r^\alpha(t), r^\beta(t)) = (\delta(t - t^\alpha_a), \delta(t - t^\beta_b)) \) to a new pair \( \xi_j(t) = (\xi^\alpha_j(t), \xi^\beta_j(t)) = (\delta(t - t^{j}_f), \delta(t - t^{j}_f - \phi_j)) \) with some desired ISI \( \phi_j \), where \( t^\alpha_a, t^\beta_b \) are the spike arrival times at the different input lines and \( t^{j}_f \) is the firing time of unit \( j \). Note that a single spike is represented by the delta function \( \delta(t) \).

The spike is transmitted via the output lines where the output spikes \( \xi^\alpha_j(t), \xi^\beta_j(t) \) of unit \( j \) have an ISI of \( \phi_j \). To ensure that unit \( j \) fires with a prespecified tolerance \( \rho \) in the timing between the input spikes, we derive the threshold value \( \theta \) such that a given unit produces an output spike pair if

\[
|t^\beta_b - t^\alpha_a - \tau_d| \leq \rho.
\]

Then the threshold \( \theta \) can be expressed as a function of \( \rho \) and the time constants \( \tau_m \) and \( \tau_s = \rho / \ln(2) \),

\[
\theta(\rho, \tau_m, \tau_s) = \frac{1}{(\tau_m / \tau_s - 1)} \left[ X_{r}^{\tau_m / (\tau_m - \tau_s)} (e^{\rho / \tau_m} + 1) - X_{r}^{\tau_m / \tau_s} (e^{\rho / \tau_s} + 1) \right],
\]

\[
X_{r} = \frac{\tau_s}{\tau_m} \left( \frac{1 + e^{\rho / \tau_m}}{1 + e^{\rho / \tau_s}} \right).
\]

Furthermore, each spiking unit in layers \( A \) and \( B \) has its own associated values of \( \tau_d \) and \( \phi_j \), these values were chosen such that each unit represents a small unique and nonoverlapping segment of the input space for a particular tolerance value \( \rho \). Therefore, changing \( \rho \) simply changes the number of units and hence the resolution of the piecewise approximation to the monotonically increasing nonlinear function \( F \) and its corresponding iterative root \( \phi \).
3. Training and results

Training the above described SNN so that the output of the hidden unit approximates the iterative root of some given nonlinear function $F$ is not trivial. Even for the case when MLPs are used, constraining the network topology such that all the weights in both subnets are pairwise identical, is essential. Furthermore, problems during training arising from local minima can seriously degrade the MLPs ability to find the iterative root, but can be overcome by better initialization.

The SNN case is no exception, we did find that initialization plays an important role in the networks ability to provide a good approximation to the iterative root. As examples, we will consider the following set of nonlinear (1D) functions, some of which the iterative root is known analytically but for others which are not.

\[
\begin{align*}
F(x) = x^2 &\quad \Rightarrow \quad \varphi(x) = \sqrt{x}, \\
F(x) = x + 1 &\quad \Rightarrow \quad \varphi(x) = x + 1/2, \\
F(x) = e^x &\quad \Rightarrow \quad \varphi(x) = ?, \\
F(x) = x^2 + 1 &\quad \Rightarrow \quad \varphi(x) = ?.
\end{align*}
\]

For these examples, the units in layers $A$ and $B$ were initialized so that each layer approximates the function $G(x) = x$, by inspection we know that the iterative root will be a function between $G(x)$ and the desired $F(x)$.

In order to train the SNN so that simultaneously the hidden unit provides a reasonable approximation to the iterative root while the output from the network approximates the desired $F$ to within some specified accuracy, two learning algorithms were used. Both algorithms are based upon backpropagating error signals from layer $B$ to layer $A$ in a pairwise fashion, but differ in respect to that the first algorithm relies on apriori knowledge while the second does not. Note that since the network must be trained to approximate both the desired $F$ and find its iterative root $\varphi$ in a piecewise manner, the number of spiking units in layers $A$ and $B$ differ, since the size of the input space for layer $B$ will be greater than layer $A$’s after training. For both algorithms, only for the units which have fired a spike are their corresponding $\phi$’s modified, since the SNN was designed so that only a single spiking unit in each respective layer fires for some given input spike pair.

In the first algorithm, the $\phi$’s of the units which fired are adjusted as follows:

**Algorithm 1.** For unit $j$ in layer $B$, training proceeds via gradient descent:

\[
\Delta \phi^B_j = \gamma (F(x_j) - \phi^B_j),
\]

\[
\phi^B_j = \phi^B_j + \Delta \phi^B_j
\]

while for unit $k$ of layer $A$,

\[
\phi^A_k = \phi^A_k + \Delta \phi^B_j + \eta (\nabla \phi^A_k - \nabla \varphi^x_j).
\]

The $\nabla \varphi$ appearing in the first algorithm represents the backward difference operator $(\nabla f_j = f_{j+1} - f_j$, representing an approximation, proportional to the derivative $f’$). In cases where apriori knowledge about the iterative root $(\nabla \varphi^*)$ is either known or can be obtained, the above penalty term ensures that the hidden unit’s output provides a good match to the iterative root. We found that this term significantly and consistently produced a better approximation to the iterative root over the domain of interest. Without such a term, there was always one or two sub-domains in which the hidden unit’s output did not satisfy (2). In the second algorithm, no apriori information about the iterative root is utilized, but a penalty term was added to provide weak coupling between the $\phi$’s of each layer. The $\phi$’s are changed according to:

**Algorithm 2.** Layer $B$ training proceeds via gradient descent, for unit $j$ we simply have

\[
\Delta \phi^B_j = \gamma (F(x_j) - \phi^B_j),
\]

\[
\phi^B_j = \phi^B_j + \Delta \phi^B_j
\]

while for unit $k$ of layer $A$,

\[
\phi^A_k = \phi^A_k + \Delta \phi^B_j \left(1 + \frac{\eta}{\gamma} (\phi^A_k - \phi^B_k)\right),
\]

where for both algorithms $\gamma$ is the learning rate, $\eta$ is a penalty factor.

In Fig. 3, we present the results of the first algorithm for the cases when $F(x)$ is either $x + 1$ or $x^2$. For these examples, the iterative root is known analytically and a value of learning rate $\gamma = 0.005$ was used. For the plots in the left column of Fig. 1, the penalty
Fig. 3. Demonstrating the effect of zero and nonzero penalty coefficients. It can be clearly seen that a nonzero value allows the network to successfully determine the iterative root across the entire range of interest. When this coefficient is zero, training fails, but the network can still find the iterative root over a finite domain smaller than the domain of interest. Blue dots and red triangles indicates the network’s approximation to the target function $F$ and iterative root $\phi$, respectively. Light blue line is the analytical root.

coefficient $\eta = 0$ while those in the right column, a nonzero value ($\eta = 0.5$) was used.

For the second algorithm, the network did find a good approximation to the iterative root of $F(x) = x^2$ (data not shown). For each respective case, $F(x) = x^2 + 1$ and $F(x) = e^x$, the calculated iterative root was compared to the one produced by the MLP, which served as a control. For each case, we found a good match for most values of $x \in [0, 2]$ except for a small portion at the beginning of this domain when $\phi(x) < F(0)$. This problem can be overcome but requires an modification to the second algorithm when $F(0) \neq 0$. For the $x^2 + 1$ case, the term $0.009/\gamma \ast (1 - \phi_k^A)$ replaced the penalty term $\eta/\gamma (\phi_k^A - \phi_k^B)$ when $\phi_k^A < F(0)$. This gave a better approximation when compared to the control. The mean squared error (MSE) was less than $10^{-4}$. While for $F(x) = e^x$, changing the penalty coefficient from $\eta$ from 0.04 to 0.002 when $\phi_k^A < F(0)$ provided a better result across the entire range when compared to control (a MSE of less than $10^{-7}$ was achieved). In Fig. 4, the results of the second algorithm (with the modification) for $F(x) = x^2 + 1$ and $F(x) = e^x$ are presented, respectively. For these simulations, like those which used first algorithm, the learning rate was $\gamma = 0.005$.

For these cases, we did confirm that these modifications are not required by choosing a better initialization, but we wanted to explore the robustness of the algorithm’s ability to simultaneously give good approximations to both the target nonlinear function and iterative root, for the initialization used to demonstrate the first algorithm (data not shown). We also confirmed that the first algorithm, based upon apriori information gained from the each respective control,
was able to find the iterative root across the range of interest with an MSE $< 10^{-6}$.

4. Discussion and future perspectives

In recent years, studies involving spiking neurons have increased in popularity. Typically, such studies usually focus on neuronal dynamics [6], firing properties of single neurons while subject to stochastic stimulation [7], or network properties of spiking neurons [8,9]. Additionally, there has been increased interest in applying networks of spiking neurons to solve engineering problems such as the XOR problem [10] and unsupervised classification [11].

Traditionally, neural network approaches based on MLPs have been employed to engineering problems but applications based upon SNNs offer an alternative framework. It must be stressed that using SNNs is not only interesting but needs to be regarded as an important advance since using SNNs to solve either engineering or neuroscience problems, increases our understanding as how SNNs behave in the “mechanistic” sense for the problem at hand but also one gains new insights as how the brain may extract, store and process information. Furthermore, using SNNs to solve more abstract types of problems such as solving functional equations should not only be interesting (although difficult) but should eventually lead to new technologies to be developed and applied to areas such as VLSI, robotics and autonomous agents. As a first step, we hope that the subject of this initial study will inspire further interest in pursuing applications based upon SNNs to solve problems of a similar nature.

In this letter, we have shown several strategies to compute the iterative root of (monotonically increasing) nonlinear functions using SNNs, highlighted some problems encountered in the work and provided some solutions and alternatives procedures. So far the SNN only provides a piecewise approximation to the iterative root, this is one drawback that needs to be overcome.

It must be emphasized that finding the iterative root of a nonlinear function by solving a functional equation is difficult. To date, there are no standard analytical techniques available to solve such functional equations and the theory dealing with them is complicated, even existence and uniqueness is not guaranteed for functions in general [2]. MLPs at least provide some form of framework which can solve such functional equations and we have shown here that SNN can also be used in the same problem domain. One immediate question does arise; if they are difficult to solve, why bother? The main reason is that using neural networks, whether artificial or spiking, for solving such equations goes beyond function approximation, but are used to find solutions to questions of higher levels of abstractness. The application can be thought of as a type of “knowledge extraction” where the iterative roots may give insights to hidden processes which may be embedded in observable data.
References


