# NEURAL APPROXIMATIONS AND THE ALGEBRA OF GRADIENTS*

Andrzej Pacut

Warsaw University of Technology
Faculty of Electronics and Information Technology
Nowowiejska 15/19, 00665 Warsaw, Poland
e-mail: pacut@ia.pw.edu.pl

*(Received September 30, 2003)*

We characterize neural networks as approximators of functions and dynamic systems. Neural approximations, leading to nonlinear minimization in highly dimensional spaces, require effective gradient calculation typically realized by gradient backpropagation. We discuss the use of gradient backpropagation for static and for dynamic systems. We also show the essential difference between the common chain rule and backpropagation, which is rarely acknowledged.

PACS numbers: 84.35.+i

## 1. Introduction

An area of neural networks can be broadly defined as the modeling inspired by biological neural mechanisms. It embraces *neural modeling*, directed into *understanding* the biological neural systems. It also includes *neural computations* inspired by biology but often having very little to do with the biological reality, with the main effort directed into solving particular computational problems better, and having typical applications in function approximation, pattern classification, global optimization *etc.*

The most popular understanding of neural networks is yet much narrower, and is related to multilayer perceptrons used as function approximators. These networks gained their popularity thanks to their numerous successful applications in a broad spectrum of problems. In this paper we characterize properties of neural networks as approximators of functions and dynamic systems. Neural approximations typically lead to nonlinear minimization in highly dimensional spaces of network parameters, realized with

* Presented at the Workshop on Applications of Neural Networks, Zakopane, Poland, May 30–June 8, 2003.

gradient minimization. Effective gradient calculation is thus of utmost importance, and is typically realized by gradient backpropagation. We discuss the use of gradient backpropagation for static as well as for dynamic systems. We also explore the essential difference between the common chain rule and backpropagation, which is rarely acknowledged.

*Notational remarks:* We typically use lower case bold, upper case bold, and lower case italic letters to denote vectors, matrices, and their elements, resp.; $T$ denotes the transposition. $\ell$ is the Lebesgue measure. $q^{-1}$ denotes the unit delay operator, namely $q^{-1}x(t) = x(t-1)$, and $\boldsymbol{q}^{-n}$ denotes the vector of the present and $n-1$ consecutive delayed values, namely $\boldsymbol{q}^{-n}x(t) = [x(t) \ \dots \ x(t-n+1)]^T$.

## 2. Function approximation ability of neural networks

### *2.1. Structure of multilayer perceptrons*

We first shortly summarize the basic entities used in neural networks [3]. The most popular artificial neuron equation is a concatenation of an affine and nonlinear transformations of the input signal $\boldsymbol{u} = [u_1 \ \cdots \ u_n]$ (Fig. 1)

$$y = g\left(b + \sum_{i=1}^{n} w_i\, u_i\right), \tag{1}$$

where $w_i$ are the *weights*, $b$ is the *bias*, and $g : \mathbb{R} \mapsto \mathbb{R}$ is called the *activation function*. Typical representatives of *continuous activation functions* are bounded, continuous, and monotone increasing nonconstant functions, called the *sigmoidal functions*. The *sigmoid*, defined as $g(u) = \frac{1}{1+\exp(a\,u)}$, and the hyperbolic tangent functions are the most common examples of the sigmoidal functions.

To create a multi-output structure out of the single-output neurons certain number of neurons receiving the same input signal are collected in parallel, thus forming the *layer* (Fig. 1)

$$y_i = g_i(b_i + \boldsymbol{w}_i^T\, \boldsymbol{u}), \quad i = 1, \dots, m\,,$$

where $\boldsymbol{u}$ is the common input to all neurons of the layer, $\boldsymbol{y} = [y_1 \ \cdots \ y_m]^T$ is layer's output, and $\boldsymbol{w}_i$ is the vector of weights of $i$-th neuron. This transformation can be compactly written as

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{u})\,, \tag{2}$$

where $\boldsymbol{g}(\boldsymbol{z}) = [g_1(z_1) \ \cdots \ g_m(z_m)]^T$ groups all activation functions, $\boldsymbol{b} = [b_1 \ \cdots b_m]^T$ is the vector of biases, and the *weight matrix* $\boldsymbol{W} = \{w_{i,j}\}_{\substack{i=1,\dots,m \\ j=1,\dots,n}}$ contains all weights.

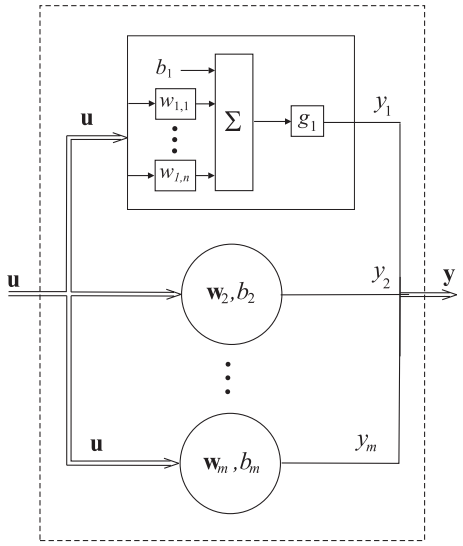


Fig. 1. A layer, with the structure of one neuron shown in details. Note that each neuron receives the same input vector $\boldsymbol{u}$.

The layers can be combined to form complex computational units called the *networks*. The simplest such construction is called the *multi-layer perceptron*, where $k-1$-th layer output $\boldsymbol{y}^{k-1}$ is used as the input to $k$-th layer, namely

$$\boldsymbol{y}^k = \boldsymbol{N}^k(\boldsymbol{y}^{k-1}), \qquad k = 1, \dots, L\,, \tag{3}$$

where $\boldsymbol{y}^0 = \boldsymbol{u}$ is the network input, $\boldsymbol{y} = \boldsymbol{y}^L$ is the network output, and $\boldsymbol{N}^k$ is the transformation performed in $k$-th layer. The signal transformation in the network can thus be presented as a concatenation of layer transformations (Fig. 2)

$$\boldsymbol{y} = \boldsymbol{N}\boldsymbol{u} = \boldsymbol{N}^L \cdots \boldsymbol{N}^2\, \boldsymbol{N}^1\, \boldsymbol{u}\,.$$

A more general combination of layers can be obtained by forming the input to $k$-th layer from the outputs of layers 1 up to $k-1$ (Fig. 3).

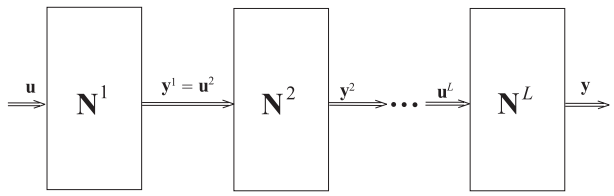


Fig. 2. Multi-layer perceptron.

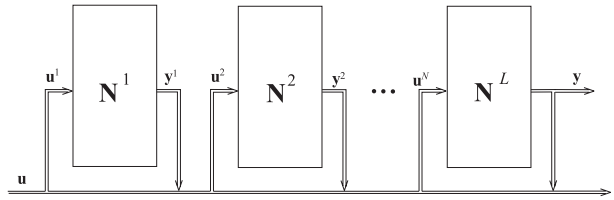


Fig. 3. Ordered system of functions.

### *2.2. Approximation problem*

The basic application of multi-layer perceptrons is *function approximation* on the basis of function samples. Loosely speaking, approximation of an unknown function $f \in \mathcal{F}$ consists of choosing a function $\widehat{f}$ from a family of approximating functions $\widehat{\mathcal{F}}$ such that $f$ and $\widehat{f}$ are as close as possible, with a given sense of two functions being close.

The question to be addressed is whether the approximating family $\widehat{\mathcal{F}}$ is rich enough to approximate every function in $\mathcal{F}$ with an arbitrary accuracy. We say that, for a given $\mathcal{F}$ and a given metric $d$, a family $\mathcal{F}$ is a *universal approximator*, if for any $\epsilon > 0$ and every $f \in \mathcal{F}$, there exists $\widehat{f} \in \widehat{\mathcal{F}}$ such that $d(f, \widehat{f}) < \epsilon$.

We will discuss the approximation properties of feedforward neural networks. While, initially, there existed only experimental evidence of approximation properties of neural networks, at present their approximation abilities are well recognized. Loosely speaking, it is known that a two-layer perceptron with linear output layer is the universal approximator for various families of functions and the related metrics, provided that the activation functions of the hidden layer fulfill certain conditions.

### *2.3. Standard approximating network*

We consider approximation of functions $f : \mathcal{U} \subset \mathbb{R}^n \mapsto \mathbb{R}$, since a generalization to multiple output function is immediate. Standard neural networks used for approximation purposes have a form of a two-layer perceptron, with linear output layer of zero bias and the hidden layer with identical activation functions $g$ (Fig. 4)

$$\boldsymbol{N}_{\boldsymbol{w}}(\boldsymbol{u}) = \sum_{i=1}^{h} w_i g \left( \sum_{j=1}^{n} \bar{w}_{i,j}\, u_j + b_i \right), \tag{4}$$

where the size $h$ of the hidden layer and $\boldsymbol{w}$ includes both the network weights and biases. We denote by $\mathcal{N}[g]$ the family of such networks, with all possible values of $h$ and $\boldsymbol{w}$, and a fixed activation function $g$.

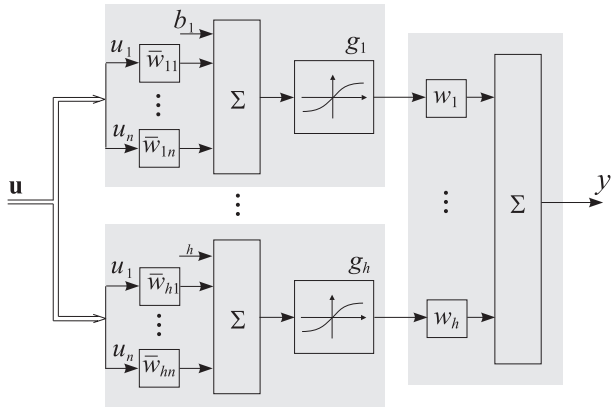


Fig. 4. Standard approximating network.

### *2.4. Approximation of continuous functions*

Consider approximation of continuous functions $f : \mathcal{U} \subset \mathbb{R}^n$ with the distance induced by the sup norm

$$\|f\|_\infty = \sup_{\boldsymbol{u}\in\mathcal{U}} |f(\boldsymbol{u})| \,. \tag{5}$$

The $\mathcal{N}[g]$ networks with sigmoidal hidden layer activation functions $g$ are called the *sigmoidal networks*. The first general approximation result was proven by Cybenko [2] who showed that for any $f$ continuous on compact domain $\mathcal{U}$ and arbitrary $\epsilon > 0$, there exists a $\mathcal{N}[g]$ network such that

$$\sup_{\boldsymbol{u}\in\mathcal{U}} |f(\boldsymbol{u}) - \widehat{f}(\boldsymbol{u})| < \epsilon$$

provided the activation function $g$ is *sigmoidal*. The condition for the activation function was subsequently relaxed. As proven by Hornik [5], $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well (in a supremum sense) approximate any continuous function on a compact subset of $\mathbb{R}^n$, provided the activation function $g : \mathbb{R} \mapsto \mathbb{R}$ is *continuous, bounded, and non-constant*. These conditions were finally weakened by Leshno *et al.* [6] who provided a necessary and sufficient condition for the activation function. A function $g : \mathbb{R} \mapsto \mathbb{R}$ is said to be *locally essentially bounded* if it is essentially bounded almost everywhere on every compact subset of $\mathbb{R}$. The distance in the set of essentially bounded functions may be induced by the norm

$$\operatorname{ess\,sup}_{\mathcal{X}} |g(x)| = \inf\big\{\delta : \ell\{x : |g(x)| \geq \delta\} = 0\big\} \,. \tag{6}$$

**Theorem 2.1.** (Leshno *et al.* [6]) *If the activation function $g$ is locally essentially bounded and the closure of its discontinuity set is of Lebesgue measure zero then $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well approximate (in the sup norm* (6) *sense) any continuous function on a compact subset of $\mathbb{R}^n$ if and only if $g$ is not a polynomial almost everywhere.*

Basically, the activation function can thus be of any type *except the polynomial*. A polynomial nonlinearity is "reproduced" in multi-layer networks as polynomial, hence this type of nonlinearity is not "rich enough" to approximate any continuous function. Note that it is not required for $g$ to be bounded, only to be locally bounded, *i.e.* bounded on every compact subset of its domain. It also does not need to be continuous, and for instance, may have a finite number of discontinuities.

The above results were extended to simultaneous approximation of a function together with its derivatives [4, 5].

### *2.5. Approximation of integrable functions*

Similar results were obtained for approximation of functions in $\mathcal{L}_p$ (integrable with $p$-th power, $1 \leq p < \infty$, on a bounded domain $\mathcal{U} \in \mathbb{R}^n$, with the distance induced by the norm

$$\|f\|_p = \left( \int\limits_{\mathcal{U}} |f(\boldsymbol{u})|^p \, d\boldsymbol{u} \right)^{1/p} . \tag{7}$$

It results from Hornik's theorem [5] that for any function in $\mathcal{L}_p$ defined on a compact set $\mathcal{U} \in \mathbb{R}^n$ and any $\epsilon > 0$ there exists a $\mathcal{N}[g]$ network for which $\|f - \widehat{f}\|_p < \epsilon$, provided the activation function $g$ is *bounded and non-constant*. These conditions for the activation function were relaxed by Leshno *et al.* [6] who also established the necessary and sufficient conditions. In fact, under the same conditions as specified in Theorem 2.1, $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well approximate (in the $\|f\|_p$ norm sense) any function integrable with $p$-th power defined on a compact set in $\mathbb{R}^n$. The above theorems can be extended to simultaneous approximations of a function continuous together with derivatives.

### *2.6. Approximation of measurable functions*

By Luzin's theorem, a real function $f$ defined on $\mathcal{U}$ is measurable if and only if for every $\epsilon > 0$ there exists a continuous function $g$ such that

$$\ell\{\boldsymbol{u} : f(\boldsymbol{u}) \neq g(\boldsymbol{u})\} < \epsilon \, . \tag{8}$$

In other words, for any measurable function on a compact domain and arbitrary $\epsilon > 0$ there exists a $\mathcal{N}[g]$ network and a compact set $\mathcal{U}_\epsilon \subset \mathcal{U}$ close to $\mathcal{U}$ in a sense that $\ell(\mathcal{U} - \mathcal{U}_\epsilon) \leq \epsilon$, for which $\sup_{\boldsymbol{u}\in\mathcal{U}_\epsilon} |f(\boldsymbol{u}) - \widehat{f}(\boldsymbol{u})| \leq \epsilon$. Consequently, if the activation function $g$ is *continuous, bounded, and non-constant* then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can *almost everywhere* arbitrarily well approximate (in the sup sense) any measurable function on a compact domain.

### *2.7. Approximation of random functions*

Assume that $\boldsymbol{u}$ is a random variable defined on a compact subset $\mathcal{U} \subset \mathbb{R}^n$ and denote its distribution by $P$. Assuming that

$$\mathcal{E}|f(\boldsymbol{u})|^p < \infty \tag{9}$$

we can take

$$\|f\|_{p,P} = \sqrt[p]{\mathcal{E}|f(\boldsymbol{u})|^p}, \qquad 1 \leq p < \infty \tag{10}$$

as the norm and define the distance accordingly. By [5], if the activation function $g$ is *bounded* and *non-constant* then for any $f$ satisfying (7) and arbitrary $\epsilon > 0$ there exists a $\mathcal{N}[g]$ network for which

$$\mathcal{E}|f(\boldsymbol{u}) - \widehat{f}(\boldsymbol{u})|^p < \epsilon\,.$$

A necessary and sufficient condition results from a theorem proven in [6]. In fact, under the conditions for the activation function $g$ specified in Theorem 2.1, $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well approximate (in the sense of (10)) any function of a continuous random variable defined on a compact set in $\mathbb{R}^n$ and satisfying (9).

### *2.8. No curse of dimensionality?*

The famous result of Barron [1] gives an upper bound on the size of the hidden layer that does not depend on the dimension of the input space, thus implying no "curse of dimensionality" for neural approximators. Assume that the distribution of $\boldsymbol{u}$ is concentrated over $\|\boldsymbol{u}\| \leq r$ and consider approximation of a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ for $\|\boldsymbol{u}\| \leq r$. Denote by $\bar{f}(\boldsymbol{\omega}) = \int \exp(j\,\boldsymbol{\omega}^T\boldsymbol{u})\, f(\boldsymbol{u})\, d\boldsymbol{u}$, $\boldsymbol{\omega} \in \mathbb{R}^n$, the ($n$-dimensional) Fourier transform of $f$. The integral

$$C_f = \int \|\boldsymbol{\omega}\|^2\, |\bar{f}(\boldsymbol{\omega})|\, d\boldsymbol{\omega} \tag{11}$$

can be regarded as a function complexity index. Barron [1] states that if the complexity index $C_f$ of the approximated function $f$ is finite then there

exists a $\mathcal{N}[g]$ network with $h$ hidden neurons such that the approximation error $d(f,\widehat{f}) = \left(\mathcal{E}\,|f-\widehat{f}|^2\right)^{1/2}$ is bounded by

$$\varepsilon \le \frac{2\,r\,C_f}{\sqrt{h}}\,. \tag{12}$$

In other words, for an approximation error bounded by $\varepsilon_0$, the number of hidden neurons

$$h \le \frac{4\,r^2\,C_f^2}{\varepsilon_0^2} \tag{13}$$

does not depend on the input dimension $n$. This result shows a computational advantage of neural networks over other approximations like polynomial approximations, splines, *etc.*, for which the required number of parameters grows exponentially with the input space dimension. Barron's result does not yet solves fully the dimensionality issue for neural networks, since the bound (13) depends on the function complexity index $C_f$ that may depend on $n$.

### *2.9. Approximations of dynamic systems*

To discuss local neural input-output approximations of nonlinear plants, we follow the approach of Levin and Narendra [8, 9]; for global representations see [7]. Consider the nonlinear plant

$$\begin{aligned} \boldsymbol{x}(t+1) &= \boldsymbol{f}\big(\boldsymbol{x}(t),\boldsymbol{u}(t)\big)\,,\\ \boldsymbol{y}(t) &= \boldsymbol{h}\big(\boldsymbol{x}(t)\big)\,. \end{aligned} \tag{14}$$

To simplify the notation we consider single-input single-output case, $\boldsymbol{u} = u$, $\boldsymbol{y} = y$. Assume the $\boldsymbol{f}$ and $\boldsymbol{h}$ are continuously twice differentiable and have stationary points at the origin, *i.e.* $\boldsymbol{f}(\boldsymbol{0},\boldsymbol{0}) = \boldsymbol{0}$, $\boldsymbol{h}(\boldsymbol{0}) = \boldsymbol{0}$. If the *linearized plant*

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}\,\boldsymbol{x}(k) + \boldsymbol{b}\,u(k)\,,\\ y(k) &= \boldsymbol{c}^T\,\boldsymbol{x}(k)\,, \end{aligned} \tag{15}$$

where

$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x},u)}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{0},0)}\,, \qquad \boldsymbol{b} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x},u)}{\partial u}\right|_{(\boldsymbol{0},0)}\,, \qquad \boldsymbol{c}^T = \left.\frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{0}} \tag{16}$$

is observable (the *observability matrix* $W_{\mathrm{o}} = [\boldsymbol{c}\ \boldsymbol{A}^T\boldsymbol{c}\ \cdots\ \boldsymbol{A}^{n-1\,T}\boldsymbol{c}]^T$ is nonsingular), then locally around the equilibrium the nonlinear system admits the

NARX (for: Nonlinear AutoRegression with eXogeneous term) representation, namely (Fig. 5)

$$y(t+1) = \boldsymbol{\psi}\big(\boldsymbol{q}^{-n}y(t),\, \boldsymbol{q}^{-n}u(t)\big)\,, \tag{17}$$

where $\boldsymbol{q}^{-n}$ denotes the tapped delay line operators (*cf.* Sec. 1). The nonlinear function $\boldsymbol{\psi}$ can be arbitrarily well approximated by a neural network $\boldsymbol{N}$, hence we obtain a neural NARX approximation of the dynamic system (14) in the form

$$y(t+1) = \boldsymbol{N}\big(\boldsymbol{q}^{-n}y(t),\, \boldsymbol{q}^{-n}u(t);\, \boldsymbol{w}\big)\,, \tag{18}$$

where $\boldsymbol{w}$ denotes a vector of the network parameters that include the weights and biases. Note that while $\boldsymbol{N}$ can approximate $\psi$ in an arbitrary region, $\psi$ approximates the nonlinear system only locally around the origin, hence the neural approximation remains local [12].

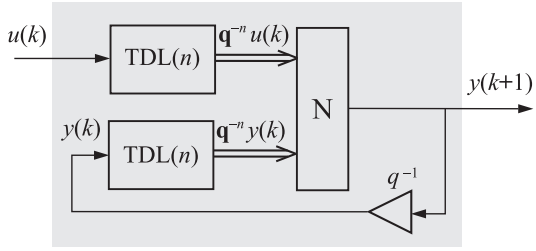


Fig. 5. NARX representation for nonlinear locally observable plants.

## 3. Algebra of gradients

Neural approximations are realized through an incremental modification of parameters to minimize a cost $Q = \sum_{t=1}^{N} q(y(t))|$ of the discrepancy between the samples of the network value $\boldsymbol{y}(t)$ and the desired output value $\boldsymbol{y}^{\mathrm{o}}(t)$, where $t$ indexes the samples. Typically, the momentary cost $q$ has a form of the quadratic index $q(y) = \|\boldsymbol{y} - \boldsymbol{y}^{\mathrm{o}}\|^2$. This problem leads to nonlinear minimization in highly dimensional space, realized with gradient minimization methods. Effective gradient calculation is thus of utmost importance for neural networks, and is typically realized by gradient backpropagation. While the gradient backpropagation was popularized by Rummelhart, Hinton, and Williams [11], it was earlier invented by Werbos and first described in his 1974 Ph.D. thesis [13]. Later, it was also independently developed by other researchers (Parker in 1985, LeCun in 1986).

We discuss the use of gradient backpropagation for static as well as for dynamic systems. We also derive the essential difference between the common chain rule and backpropagation, which is rarely acknowledged.

### *3.1. Layered systems of functions*

A family of functions

$$x_k = f_k(x_{k-1}), \qquad k = 1, \ldots, n \tag{19}$$

with appropriately defined domains, will be called the *layered system of functions*, Fig. 2. A typical example of the layered system is the multilayer perceptron (3). We discuss algorithms of calculation of the derivative $\dfrac{dx_n}{dx_0}$. Denote $f'_k = \dfrac{df_k}{dx_{k-1}}$ for $i = 1, \ldots, n$, and for $x_0, \ldots x_{\ell-1}$ fixed, $\ell < k$, denote $x'_{k|\ell} = \dfrac{dx_k}{dx_\ell}$. Obviously, by the chain rule

$$\frac{dx_n}{dx_0} = f'_n(x_{n-1})\, f'_{n-1}(x_{n-2}) \,\ldots\, f'_2(x_1)\, f'_1(x_0)\,. \tag{20}$$

The derivative (20) can be calculated recursively in two ways, namely with the forward propagation algorithm and the backpropagation algorithm.

### *3.2. Forward vs backward gradient propagation*

The forward propagation (FP) algorithm is just a direct implementation of the traditional *chain rule* and results from grouping the intermediate results, illustrated as

$$x'_{n|0} = \frac{dx_n}{dx_0} = f'_n(x_{n-1})\ \underbrace{f'_{n-1}(x_{n-2})\ \ldots\ \underbrace{f'_2(x_1)\ \underbrace{f'_1(x_0)}_{x'_{1|0}}}_{x'_{2|0}}}_{x'_{n-1|0}}\,. \tag{21}$$

The algorithm resulting from the above approach can be written in the form

$$\begin{aligned} x'_{0|0} &= 1\,, \\ x'_{k|0} &= f'_k(x_{k-1})\, x'_{k-1|0}\,, \\ x_k &= f_k(x_{k-1})\,, \qquad\qquad k = 1, \ldots, n\,. \end{aligned} \tag{22}$$

Note that both the derivatives as well as the function values are calculated in a single pass of recursion, with $k = 1, \ldots, n$.

The second method of derivative calculations, which is called the *back-propagation* (BP) results from a different grouping of terms in (20), according to

$$x'_{n|0} = \frac{dx_n}{dx_0} = \underbrace{\underbrace{\underbrace{f'_n(x_{n-1})}_{x'_{n|n-1}}\, f'_{n-1}(x_{n-2})}_{x'_{n|n-2}} \,\ldots\, f'_2(x_1)}_{x'_{n|1}}\, f'_1(x_0)\,. \tag{23}$$

This leads to an algorithm that can be realized in two passes: the *forward pass* of arguments calculation

$$x_k = f_k(x_{k-1})\,, \qquad k = 1, \ldots, n$$

and the *backward pass* of the derivative calculation (output-to-input), namely

$$\begin{aligned} x'_{n|n} &= 1\,, \\ x'_{n|\ell} &= x'_{n|\ell+1}\, f'_{\ell+1}(x_\ell)\,, \qquad \ell = n-1, \ldots, 0\,. \end{aligned} \tag{24}$$

### *3.3. Ordered system of functions*

It is useful to consider a more general construction, namely the *ordered family* of functions, defines as a family of functions $(f_1, \ldots, f_n)$

$$x_k = f_k(x_0, \ldots, x_{k-1}) \tag{25}$$

with appropriately defined domains. One may represent the values calculated by each function by vertices of an ordered graph which we will call the *influence graph*. Functions of the system can then be represented by directed edges which join the argument vertices (the ones corresponding to the function arguments) with the value vertex (the one corresponding to the function value). The graph representing the ordered function is without loops. The general feedforward networks (Fig. 3) make a typical examples of ordered functions.

Denote

$$f'_{k|\ell}(x_0, \ldots, x_{k-1}) = \frac{\partial}{\partial x_\ell}\, f_k(x_0, \ldots, x_{k-1}), \quad 0 \le \ell < k \tag{26}$$

and for the first $\ell$ variables $x_0, \ldots, x_{\ell-1}$, $\ell < k$, fixed define

$$x'_{k|\ell} = \frac{dx_k}{dx_\ell}\,. \tag{27}$$

Note that in calculation of $x'_{k|\ell}$ we treat $x_\ell$ as the only independent variable. One may prove that the following equalities hold and are equivalent

$$x'_{k|\ell} = f'_{k|\ell} + \sum_{i=\ell+1}^{k-1} f'_{k|i}\, x'_{i|\ell} \quad 0 \le \ell < k \le n\,, \tag{28}$$

$$x'_{k|\ell} = f'_{k|\ell} + \sum_{j=\ell+1}^{k-1} x'_{k|j}\, f'_{j|\ell} \quad 0 \le \ell < k \le n\,, \tag{29}$$

where we treat a sum as zero if its lower summation index exceeds the upper one. Here (28) represents the forward propagation algorithms for all subsystems of equations, and setting $\ell = 0$ specializes the equations to $x'_{n|0} = \dfrac{dx_n}{dx_0}$. Similarly, (29) represents the *backpropagation algorithms* for all possible subsystems of equations, and setting $k = n$ enables to calculate $x'_{n|0} = \dfrac{dx_n}{dx_0}$.

As an illustration, consider the system of equations $x_3 = f_3(x_0, x_1, x_2)$, $x_2 = f_2(x_0, x_1)$, $x_1 = f_1(x_0)$ and assume that $f_1, f_2, f_3$ are differentiable. With the FP we obtain

$$\begin{aligned} x'_{3|0} &= f'_{3|0} + f'_{3|1}\, x'_{1|0} + f'_{3|2}\, x'_{2|0}\,,\\ x'_{2|0} &= f'_{2|0} + f'_{2|1}\, x'_{1|0}\,,\\ x'_{1|0} &= f'_{1|0}\,, \end{aligned}$$

while the BP formulas have a form

$$\begin{aligned} x'_{3|0} &= f'_{3|0} + x'_{3|2}\, f'_{2|0} + x'_{3|1}\, f'_{1|0}\,,\\ x'_{3|1} &= f'_{3|1} + x'_{3|2}\, f'_{2|1}\,,\\ x'_{3|2} &= f'_{3|2}\,. \end{aligned}$$

Certainly, both algorithms lead to $x'_{3|0} = f'_{3|0} + f'_{3|1}\, f'_{1|0} + f'_{3|2}\, f'_{2|0} + f'_{3|2}\, f'_{2|1}\, f'_{1|0}$, just the terms are grouped differently. Note that different derivatives of $x$ are used in both formulas. In the FP we use $x'_{i|0}$ for $i = 1, 2, 3$, while in the BP we use $x'_{3|j}$ for $j = 2, 1, 0$.

The apparent difference between the two algorithms consists in the order of calculations. More important yet are the intermediate derivatives calculated by both algorithms. In the forward propagation case, the intermediate results consist of the derivatives of consecutive variables *with respect to the same target independent variable*. In the case of BP, the intermediate results

consist of the derivatives *of the same target variable* with respect to consecutive intermediate variables, Fig. 6. In network calculations, the derivatives of the same error index are to be calculated with respect to all weights. The derivatives calculated intermediately in the BP algorithm are of direct use, while the intermediate derivatives calculated in forward propagation are useless. These very property stands for the effectiveness of the BP in neural networks.

Note yet that the BP equations are easiest to *derive* "in the direction of the influence graph", while the BP *calculations* are done in the reverse direction.

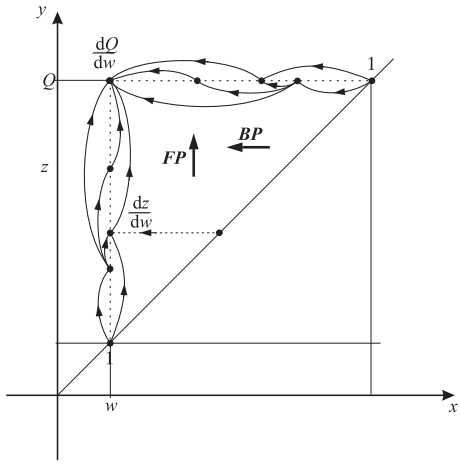


Fig. 6. Backpropagation, forward propagation and mixed modes. Each point $(x, y)$ corresponds to the derivative $\dfrac{dy}{dx}$. Mixed mode illustrated here uses the BP first (horizontal dashed line) and the FP afterwards.

A matrix form put another light on the difference between the FP and BP. Define two $n+1$ by $n+1$ matrices, $\boldsymbol{F'}$ and $\boldsymbol{X'}$, by using (26), (27) for $k > \ell$, and setting $f'_{k|\ell} = 0 \quad \text{for } k \le \ell$, $x'_{k|\ell} = \begin{cases} 1 & \text{for } k = \ell \\ 0 & \text{for } k < \ell \end{cases}$. Note that for a layered system only the sub-diagonal of $\boldsymbol{F'}$ is non-zero. One may prove [10] that the following equalities are equivalent to (28), (29)

$$\text{Chain rule} \quad (\boldsymbol{F'} - I)\,\boldsymbol{X'} = -I\,, \tag{30}$$

$$\text{Backpropagation} \quad \boldsymbol{X'}\,(\boldsymbol{F'} - I) = -I\,. \tag{31}$$

Again, (30) represents the FP rules and (31) represents the BP rules for all possible subsystems.

### *3.4. Local forms*

In general, for any two variables $u = x_k$, $z = x_\ell$, $\ell < k$, the FP can be compactly written as

$$\frac{dz}{du} = \frac{\partial f_z}{\partial u} + \sum_x \frac{\partial f_z}{\partial x}\,\frac{dx}{du}\,, \tag{32}$$

where $f_z = f_\ell$ is a function which defines $z$, and the sum extends over all variables $x$ *which directly influence* $z$ through $f_z$ (*i.e.*, the arguments of $f_z$), Fig. 7. On the other hand, for any two variables $u = x_k$, $z = x_\ell$, $\ell < k$, the BP formula has the form

$$\frac{dz}{du} = \frac{\partial f_z}{\partial u} + \sum_x \frac{dz}{dx}\,\frac{df_x}{du}\,, \tag{33}$$

where $f_x$ denotes the function that defines $x$, and the summation extends over all terms $x$ that *are directly influenced* by $u$ (*i.e.*, all functions in the system whose one of arguments is $u$), Fig. 8.

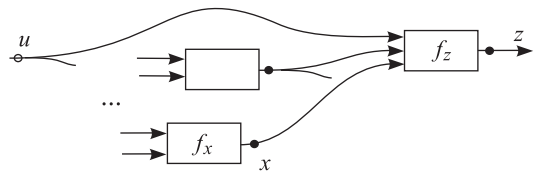


Fig. 7. Forward propagation for ordered systems. The summation in (32) extends over all variables $x$ *that directly influence* $z$ through $f_z$ (*i.e.*, the arguments of $f_z$).

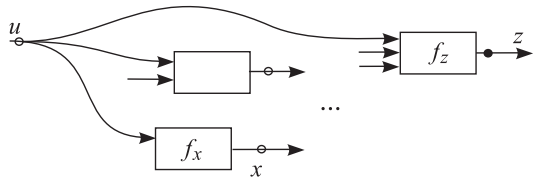


Fig. 8. Backpropagation for ordered systems. The summation in (33) extends over all terms $x$ that *are directly influenced* by $u$ (*i.e.*, all functions in the system whose one of arguments is $u$).

## 4. Implementations of gradient backpropagation

### *4.1. Gradients in multi-layer perceptron networks*

Consider a $L$-layer network, with $\ell$-th layer equations

$$\begin{aligned} z_i^\ell &= \sum_{j=0}^{n^\ell} w_{i,j}^\ell \, y_j^{\ell-1}\,, \\ y_i^\ell &= g_i^\ell(z_i^\ell)\,, \qquad i = 1, \ldots, m^\ell\,, \end{aligned} \tag{34}$$

the network outputs $y_j = y_j^{(L)}$, and the momentary cost $q = \rho(y_1^L, \ldots, y_m^L)$. It is easy to see that the influence graph (Fig. 9) corresponds to the following chain of calculations

$$\left.\begin{aligned} \frac{dq}{dw_{i,j}^{\ell-1}} &= \frac{dq}{dz_i^{\ell-1}}\, u_j^{\ell-1} \\ \frac{dq}{dz_i^{\ell-1}} &= \frac{dq}{dy_i^{\ell-1}}\, g_i^{\ell-1\prime}(z_i^{\ell-1}) \end{aligned}\right\} \text{ weights in layer } \ell-1 \text{ to output}$$

$$\left.\begin{aligned} \frac{dq}{dy_i^{\ell-1}} &= \textstyle\sum_{k=1}^{m^\ell} \frac{dq}{dz_k^\ell}\, w_{k,i}^\ell \\ \frac{dq}{dz_k^\ell} &= \frac{dq}{dy_k^\ell}\, g_k^{\ell\prime}(z_k^\ell) \end{aligned}\right\} \text{ through layer } \ell$$

$$\left.\frac{dq}{dy_p^L} = \frac{\partial \rho(\boldsymbol{y})}{\partial y_p^L} = \rho_p'(\boldsymbol{y})\right\} \text{cost}\,.$$

Consequently, the BP to a weight in $p$-th layer consists of the BP through the cost index (35), then the BP through consecutive layers $\ell = L, \ldots, p+1$,

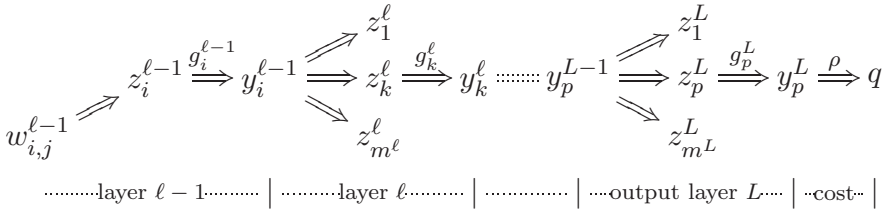


Fig. 9. The influence graph for a multi-layer network.

(36), and the BP to the weight in $p$-th layer, namely

$$\frac{dq}{dy_p} = \frac{\partial \rho(\boldsymbol{y})}{\partial y_p} = \rho'_p(\boldsymbol{y})\,, \tag{35}$$

$$\frac{dq}{dy_i^{\ell-1}} = \sum_{k=1}^{m^\ell} \frac{dq}{dy_k^\ell}\, g_k^{\ell\prime}(z_k^\ell)\, w_{k,i}^\ell\,, \tag{36}$$

$$\frac{dq}{dw_{i,j}^p} = \frac{dq}{dy_i^p}\, g_i^{p\prime}(z_i^p)\, u_j^p\,. \tag{37}$$

A particularly simple form of the BP is obtained for sigmoid activation function $y = g(x) = \frac{1}{1+\mathrm{e}^{-\alpha x}}$ for which $g'(x) = -\alpha\, y\,(1-y)$.

The equivalent matrix notation has a compact form

$$\begin{aligned}
\frac{dq}{d\boldsymbol{y}} &= \frac{\partial \rho(\boldsymbol{y})}{\partial \boldsymbol{y}} = \rho'(\boldsymbol{y})\,,\\
\frac{dq}{d\boldsymbol{y}^{\ell-1}} &= \boldsymbol{W}^{\ell T}\Big(\frac{dq}{d\boldsymbol{y}^\ell} \odot \boldsymbol{g}^{\ell\prime}(\boldsymbol{z}^\ell)\Big), \qquad \ell = L, \ldots, p+1\,,\\
\frac{dq}{d\boldsymbol{W}^p} &= \Big(\frac{dq}{d\boldsymbol{y}^p} \odot \boldsymbol{g}^{p\prime}(\boldsymbol{z}^p)\Big)\, \boldsymbol{u}^{pT}\,,
\end{aligned}$$

where $\odot$ denotes the term-by-term multiplication of vectors, namely $\boldsymbol{x} \odot \boldsymbol{y} = \begin{bmatrix} x_1 y_1 & \ldots x_n y_n \end{bmatrix}^T$. It is convenient to introduce an operator $\mathcal{B}_{\boldsymbol{W}}$ that transforms $\dfrac{d}{d\boldsymbol{y}}$ into the derivative with respect to weights in the same layer, thus *backpropagating to weights* from layer's output, namely (Fig. 10)

$$\frac{d}{d\boldsymbol{W}} = \mathcal{B}_{\boldsymbol{W}}\left(\frac{d}{d\boldsymbol{y}}\right) = \left(\frac{d}{d\boldsymbol{y}} \odot \boldsymbol{g}'(\boldsymbol{z})\right)\, \boldsymbol{u}^T\,. \tag{38}$$

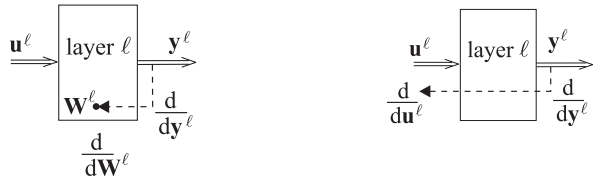


Fig. 10. BP to weights in a layer (left), BP through the layer (right).

Another operator we introduce, $\mathcal{B}_{\boldsymbol{u}}$, transforms $\dfrac{d}{d\boldsymbol{y}}$ into the derivative with respect to the inputs of the layer, thus *backpropagating through the layer*, namely

$$\frac{d}{d\boldsymbol{u}} = \mathcal{B}_{\boldsymbol{u}}(\nabla_{\boldsymbol{y}}) = \boldsymbol{W}^T\Big(\frac{d}{d\boldsymbol{y}} \odot \boldsymbol{g}'(\boldsymbol{z})\Big)\,. \tag{39}$$

Using the BP operators, we can write the BP through the network to the weights in a form (Fig. 11)

$$\frac{d}{d\boldsymbol{W}^{\ell}} = \mathcal{B}_{\boldsymbol{W}^{\ell}}\, \mathcal{B}_{\boldsymbol{u}^{\ell+1}} \ldots \mathcal{B}_{\boldsymbol{u}^{L}} \frac{d}{d\boldsymbol{y}^{L}} \tag{40}$$

and the BP through the network to its inputs as

$$\frac{d}{d\boldsymbol{u}^{1}} = \mathcal{B}_{\boldsymbol{u}^{1}}\, \mathcal{B}_{\boldsymbol{u}^{2}} \ldots \mathcal{B}_{\boldsymbol{u}^{L}} \frac{d}{d\boldsymbol{y}^{L}}\,. \tag{41}$$

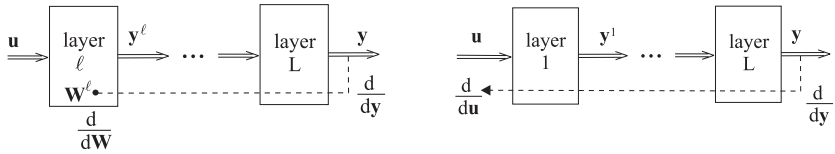


Fig. 11. BP to the weights of a network (left), BP through the network (right).

### *4.2. Second derivatives*

Calculation of second derivatives, required in some algorithms, is more complex, with or without the BP. We recall a little known way to approximate second derivatives proposed by Werbos [14] (Fig. 12). Suppose we want to approximate Hessian of the function $y^{o} = f(\boldsymbol{u})$. The first (single output) network $\boldsymbol{N}$ calculates an approximation of $f$, namely $\boldsymbol{N}(\boldsymbol{u}) = \widehat{f}(\boldsymbol{u})$, and the BP is used to calculate the gradients of $\widehat{f}$. The approximated gradient values serve in turn as the desired values to another network $\boldsymbol{N}^{1}$, which thus approximates the gradient of the approximated function $\widehat{f}'$, namely $\boldsymbol{N}^{1}(\boldsymbol{u}) = \widehat{\widehat{f}}'$. The BP through this network approximates Hessian $\widehat{\widehat{f}''}$.

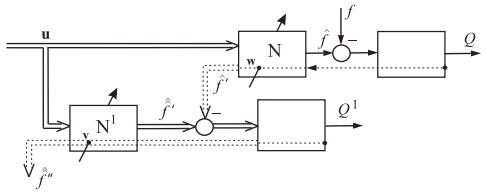


Fig. 12. Using two networks to approximate the second derivatives.

*4.3. Gradient calculations in nonlinear dynamic systems*

Suppose that for a dynamic plant (14) both $\boldsymbol{f}$ and $\boldsymbol{g}$ are approximated by the state network $\widehat{\boldsymbol{f}}$ and the observation network $\widehat{\boldsymbol{h}}$, namely

$$\begin{aligned} \boldsymbol{x}(t+1) &= \widehat{\boldsymbol{f}}\big(\boldsymbol{x}(t), \boldsymbol{u}(t);\, \boldsymbol{w}\big)\,, \\ \boldsymbol{y}(t) &= \widehat{\boldsymbol{h}}\big(\boldsymbol{x}(t);\, \boldsymbol{v}\big)\,, \end{aligned} \tag{42}$$

where $\boldsymbol{w}$ and $\boldsymbol{v}$ denote the parameter vectors of both networks. The networks are to minimize the cost $Q = \sum_{t=1}^{N} \|\boldsymbol{y}(t) - \boldsymbol{y}^{\mathrm{o}}(t)\|^2$ of the discrepancy between the modeled plant output $\boldsymbol{y}$ and the desired output $\boldsymbol{y}^{\mathrm{o}}$. To avoid cluttering of formulas we skip the intermediate arguments of functions, *e.g.*, we write $\widehat{\boldsymbol{f}}(t)$ instead of $\widehat{\boldsymbol{f}}\big(\boldsymbol{x}(t), \boldsymbol{u}(t);\, \boldsymbol{w}\big)$.

For the state network, to calculate $\dfrac{dQ}{dw}$, where $w$ is any element of $\boldsymbol{w}$, we notice first that $w$ influences all state coordinates at every moment $t$. Consequently

$$\frac{dQ}{dw} = \sum_{t=1}^{N} \sum_{j=1}^{n} \frac{dQ}{dx_j(t)} \frac{\partial \widehat{f}_j(t)}{\partial w}\,, \tag{43}$$

where the partial derivative can itself be calculated with the use of the BP once the structure of the network $\widehat{\boldsymbol{f}}$ is known. Since every coordinate $x_i(t)$ of the state vector influences every coordinate of the present output vector $y_j(t)$ through the observation equation, and every coordinate of the state at the next moment (except at the last moment $t = N$) through the state equation, we have

$$\begin{aligned} \frac{dQ}{dx_i(t)} &= \sum_{j=1}^{p} \frac{dQ}{dy_j(t)} \frac{\partial \widehat{h}_j(t)}{\partial x_i(t)} + \sum_{j=1}^{n} \frac{dQ}{dx_j(t+1)} \frac{\partial \widehat{f}_j(t)}{\partial x_i(t)}\,, \quad \text{for } t < N\,, \\ \frac{dQ}{dx_i(N)} &= \sum_{j=1}^{p} \frac{dQ}{dy_j(N)} \frac{\partial \widehat{h}_j(N)}{\partial x_i(N)}\,, \end{aligned} \tag{44}$$

where the partial derivatives can themselves be calculated by the BP once the structures of the networks $\widehat{\boldsymbol{f}}$ and $\widehat{\boldsymbol{h}}$ are known. Finally, every output coordinate directly influences the cost index, hence

$$\frac{dQ}{dy_i(t)} = y_i(t) - y_i^{\mathrm{o}}(t)\,. \tag{45}$$

Calculation of the derivative $\dfrac{dQ}{dv}$, where $v$ is any element of $\boldsymbol{v}$ is immediate

$$\frac{dQ}{dv} = \sum_{t=1}^{N}\sum_{j=1}^{n}\big(y_i(t) - y_i^{\mathrm{o}}(t)\big)\,\frac{\partial \widehat{h}_j(t)}{\partial v} \tag{46}$$

when the derivatives $\dfrac{\partial \widehat{h}_j(t)}{\partial v}$ can again be calculated with the use of the BP once the structure of $\widehat{\boldsymbol{h}}$ is known. Introducing the sensitivity vectors $Q_{\boldsymbol{x}}(t) = \left[\dfrac{dQ}{dx_1(t)} \quad \cdots \quad \dfrac{dQ}{dx_n(t)}\right]^T$ we may rewrite all the above equations in a form of linear recurrent equations with the time reversed, namely

$$\begin{aligned} Q_{\boldsymbol{x}}(t) &= \widehat{\boldsymbol{f}}_{\boldsymbol{x}}(t)^T\, Q_{\boldsymbol{x}}(t{+}1) + \widehat{\boldsymbol{h}}_{\boldsymbol{x}}(t)^T\big(\boldsymbol{y}(t) - \boldsymbol{y}^{\mathrm{o}}(t)\big)\,,\\ Q_{\boldsymbol{w}}(t) &= Q_{\boldsymbol{w}}(t{+}1) + \widehat{\boldsymbol{f}}_{\boldsymbol{w}}(t)^T Q_{\boldsymbol{x}}(t)\,,\\ Q_{\boldsymbol{v}}(t) &= Q_{\boldsymbol{v}}(t{+}1) + \widehat{\boldsymbol{h}}_{\boldsymbol{v}}(t)^T\big(\boldsymbol{y}(t) - \boldsymbol{y}^{\mathrm{o}}(t)\big) \end{aligned} \tag{47}$$

for $t = N, \ldots, 1$, with the initial conditions

$$Q_{\boldsymbol{x}}(N{+}1) = \mathbf{0}\,, \qquad Q_{\boldsymbol{w}}(N{+}1) = \mathbf{0}\,, \qquad Q_{\boldsymbol{v}}(N{+}1) = \mathbf{0} \tag{48}$$

and with $\dfrac{dQ}{dw}$ and $\dfrac{dQ}{dv}$ as the elements of vectors $Q_{\boldsymbol{w}}(1)$, $Q_{\boldsymbol{v}}(1)$, respectively.

### *4.4. NARX representation*

Consider the neural NARX representation (18) of a single-input single-output system. We calculate the gradients needed to minimize the cost $Q = \sum_{t=1}^{N} q(y(t))$ of discrepancy between the modeled plant output $y$ and the desired output. Since $w$ influences network's output at all moments $t$ then

$$\frac{dQ}{dw} = \sum_{t=1}^{N} \frac{dQ}{dy(t)}\,\frac{\partial \boldsymbol{N}(t)}{\partial w}\,.$$

Now, $y(t)$ influences $y(t+1), \ldots, y(t+n)$ (through TDL and the network) and the cost, hence

$$\frac{dQ}{dy(t)} = \sum_{s=t+1}^{t+n} \frac{dQ}{dy(s)}\,\frac{\partial \boldsymbol{N}(s)}{\partial y(t)} + q'(t)\,.$$

This involves calculation of $\dfrac{dQ}{dy(t)}$ *in the reversed time* for $t = N, \ldots, 1$, with boundary conditions $\dfrac{dQ}{dy(s)} = 0$ for $s = N+1, \ldots, N+n$. Calculation of $\dfrac{\partial \boldsymbol{N}(t)}{\partial w}$ may itself employ the BP through the network.

## 5. Summary

We discussed chosen properties of neural networks as approximators, considering both the function approximation problems as well as dynamic system approximation problems. Since neural approximations lead to nonlinear minimization in highly dimensional spaces of their parameters, it is necessary to be able to effectively calculate gradients with respect to network parameters, typically with the use of gradient backpropagation. We discuss applications of gradient backpropagation in both static and dynamic systems. We also investigate the essential differences between the common chain rule and backpropagation. We show that both methods may be reduced to matrix equations with a different order of multiplication of commutative matrices.

The problems discussed in this paper are intended to present the multilayer perceptrons as valid function approximators, whose properties may exceed those of other nonlinear approximators. Treating the multilayer perceptrons as nonlinear counterparts of linear least-square methods may help in proper evaluation of the predicted effects of neural network applications.

## REFERENCES

[1] A. Barron, *IEEE Trans. Information Theory* **3**, 930 (1993).

[2] G. Cybenko, *Mathematics of Control, Signals, and Systems* vol. 2, 1989, pp. 303–314.

[3] S. Haykin, *Neural Networks. A Comprehensive Foundation. Second Edition*, Prentice Hall, Inc., Upper Saddle River, New Jersey, USA 1999.

[4] K. Hornik, M. Stinchcombe, H. White, *Neural Networks* **3**, 551 (1990).

[5] K. Hornik, *Neural Networks* **4**, 251 (1991).

[6] M. Leshno, V. Lin, A. Pinkus, S. Schocken, *Neural Networks* **6**, 861 (1993).

[7] A.U. Levin, K.S. Narendra, *Int. J. Control* **61**, 533 (1995).

[8] A.U. Levin, K.S. Narendra, *IEEE Trans. Neural Networks* **4**, 192 (1993).

[9] K.S. Narendra, *Proc. IEEE* **84**, 1385 (1996).

[10] A. Pacut, *Proc. of the 2002 International Joint Conference on Neural Networks IJCNN'02*, Honolulu, HA, IEEE Press, Piscataway, NJ 2002, pp. 530–534.

[11] D.E. Rummelhart, G.E. Hinton, R.J. Williams, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, eds. D.E. Rummelhart, J.L. McClelland, vol. 1, Chap. 8, MIT Press, Cambridge, MA 1986.

[12] I.W. Sandberg, *IEEE Trans. Circuits and Systems* **38**, 564 (1991).

[13] P.J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Ph.D. Thesis, Harvard University, Cambridge, MA, 1974.

[14] P.J. Werbos, `http://xxx.lanl.gov/html/adap-org/9810001`, 1998.