Polska Akademia Nauk • Instytut Badań Systemowych

AUTOMATYKA STEROWANIE ZARZĄDZANIE

Książka jubileuszowa
z okazji
70-lecia urodzin

PROFESORA KAZIMIERZA MAŃCZAKA

pod redakcją
Jakuba Gutenbauma

Polska Akademia Nauk • Instytut Badań Systemowych

AUTOMATYKA
STEROWANIE
ZARZĄDZANIE

Książka jubileuszowa
z okazji
70-lecia urodzin

PROFESORA KAZIMIERZA MAŃCZAKA

pod redakcją
Jakuba Gutenbauma

Warszawa 2002

Książka jubileuszowa z okazji
70-lecia urodzin
Profesora Kazimierza MAŃCZAKA

Redaktor
prof. dr hab. inż. Jakub Gutenbaum

ISBN 83-85847-78-2

Wydawca: Instytut Badań Systemowych PAN
ul. Newelska 6 01-447 Warszawa
http://www.ibspan.waw.pl

Opracowanie składopisu: Anna Gostyńska, Jadwiga Hartman

Druk: KOMO-GRAF, Warszawa
nakład 200 egz., 34 ark. wyd., 31 ark. druk.

ADJOINT MULTILAYER NEURAL NETWORKS

Maciej Krawczak
Systems Research Institute
Polish Academy of Sciences
University of Information Technology and Management
Newelska 6, 01-447 Warsaw, Poland

Abstract: *In the artificial neural networks literature little attention has been given to consideration of neural networks from the point of view of graph theory. Examination of neural networks as flow graphs gives very interesting and new properties of the neural networks' learning process. The approach is based on tools used in electric circuits when Kirchhoff's laws may not be valid. The analysis leads to the proper equations of the backpropagation algorithm, but in a much simpler manner. The graph methodology incorporates the reciprocal graphs in which signals flow in opposite directions. Neural networks in which signals flow in opposite direction are called the adjoint neural networks.*

Keywords: *multilayer neural networks, learning from examples, flow graphs.*

1. Introduction

Learning of multilayer neural networks is usually performed by the backpropagation algorithm, often with some modifications. The algorithm is a gradient descent one and has a long history. The basic version of the algorithm is well known from Rumelhart and McClelland (1986), and it was derived by applying repeatedly the chain rule expansions backward through the network. There are several researchers who claim the right to this algorithm. We would like to mention here the work by Werbos (1974), who obtained similar results by applying ordered partial derivatives, and by Dreyfus (1990), who obtained similar results relating the gradient theory of flight paths from the late 1950s. Considering the learning process of multilayer neural networks as a particular multistage optimal control problem, Krawczak (1994, 1999, 2000) obtained backpropagation-like equations.

In the very rich artificial neural networks literature little attention has been given to consideration of neural networks from the point of view of graph theory. Examination of neural networks *as flow graphs* gives very interesting and new properties of the neural networks learning process. The approach is based on the Tellegen's theorem (Chua and Lin 1975) used in the electric circuits when Kirchhoff's laws may not be valid. The analysis leads to the proper equations of the backpropagation algorithm, but in a much simpler manner. Here we use terminology adopted from the optimization or optimal control theory (Bryson and Ho 1969), and such neural networks in which signals flow in opposite direction are called the *adjoint neural networks.*

In this paper we present a flow graph methodology for representing neural networks, and then the adjoint neural networks for representing learning algorithms based on gradient methods. The backpropagation algorithm is considered in details, while the applied notation allows also for treating multilayer networks with possible feedback signals. Within the backpropagation algorithm we emphasize the forward propagation of the inputs through the neural networks and the backward propagation of proper gradients through the adjoint neural networks. Construction of the adjoint neural networks yields directly the formulae of the considered learning algorithm.

2. Transformation of a Neural Network into a Flow Graph

Since the pioneering work of McCulloch and Pitts (Zurada 1992) a model of an artificial neuron is a very simple processing unit, Figure 1,

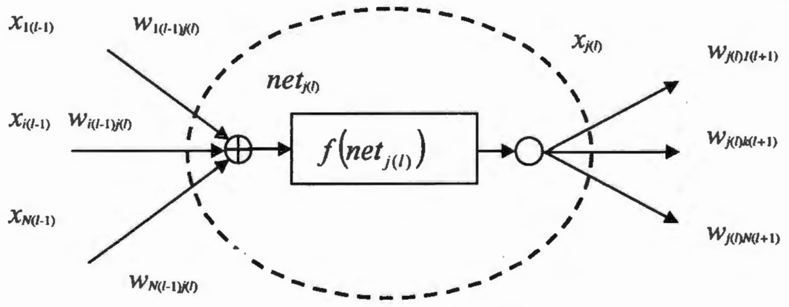

Fig.1. An elementary model of a neuron.

which has a number of inputs x_i, say N, each input being weighted with an appropriate weight w_{ij}, $i = 1,2,...,N$. The sum of the weighted inputs and the bias (included in the inputs) form at the summation point ⊕ the proper input

$$net_j = \sum_{i=1}^{N} w_{ij}\, x_i = \sum_{i=1}^{N} y_{ij} \,, \tag{1}$$

to the activation function $f_j(net_j)$. Here we consider the differentiable activation functions for generation of outputs from inputs of the neurons. In the model considered an additional element, depicted by O, is included, corresponding to a junction point. Generally, the existence of the junction point in a neuron has been tacitly assumed. Figure 1 shows an extended notation of indices, namely we indicate the position of each neuron in the whole network. For example the weight $w_{i(l-1)\,j(l)}$ indicates the connection between the neuron i belonging to the $(l-1)$-st layer and the neuron j from the (l)-th layer.

In this way three main elements of a neuron can be distinguished, one part is a summation point, the second is an activation function which transmits the effect of summation, and the third is a junction point spreading the value resulting from the activation function output to neurons of the next layer.

Let us rearrange the neuron's elements in the following way:

- remove the activation function to the outside of the neuron,
- the removed activation functions are shifted to each of the connections between the considered neuron and all neurons of the next layer, becoming thereby the transmittances between neurons,
- the connection between neurons are still weighted,
- the summation point and the junction point make up a *node*,
- the neural network with the rearranged neurons becomes a *flow graph.*

The above rearrangement is pictured in Figure 2.

Now, Equation 1 can be rewritten in the following way:

$$net_{j(l)} = \sum_{i=1}^{N} w_{i(l-1)j(l)}\, x_{i(l-1)} = \\ = \sum_{i=1}^{N} w_{i(l-1)j(l)}\, f_{i(l-1)}\left(net_{i(l-1)}\right) = \sum_{i=1}^{N} y_{i(l-1)j(l)} \tag{2}$$

and description of any separate edge takes the following form:

$$y_{i(l-1)j(l)} = w_{i(l-1)j(l)}\, x_{i(l-1)} = w_{i(l-1)j(l)}\, f_{i(l-1)}\left(net_{i(l-1)}\right). \tag{3}$$

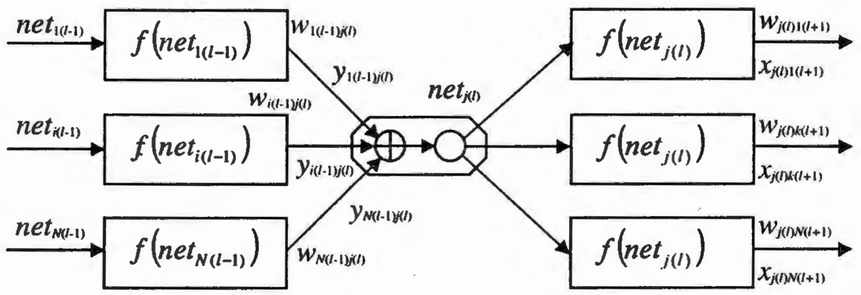

Fig.2. The rearranged neuron with its vicinity as a segment of a flow graph.

Now let us show an example of a simple neural network with one hidden layer, Figure 3. After using the rearranging procedure described above the same neural network can be considered as a flow graph of Figure 4.

When comparing these two pictures we can notice that the architectures of the neural network and the flow graph are exactly the same, in the position of each neuron a graph node has appeared, while to each connection an activation function has been added as a transmittance. In Figure 4 two kinds of reduced nodes appear:

- input nodes are reduced to junction points,
- output nodes are reduced to summation points.

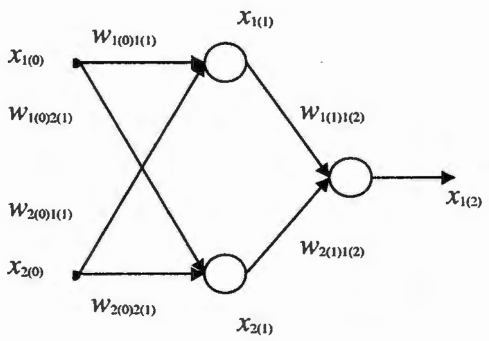

Fig.3. A simple two-layer neural network.

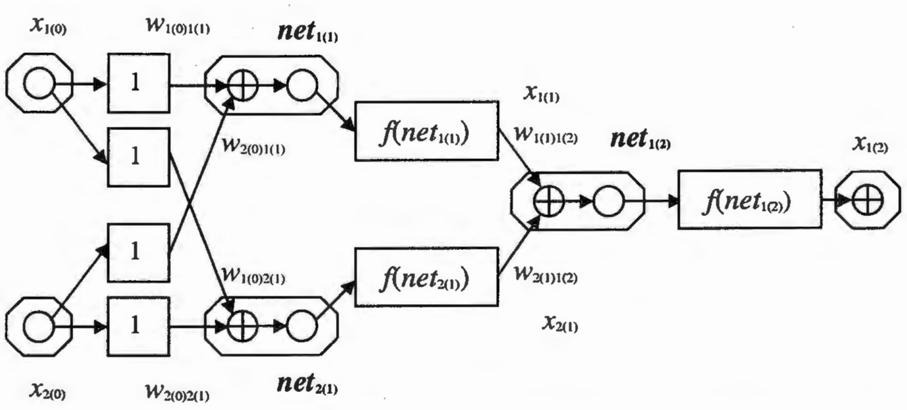

Fig.4. A flow graph corresponding to the two-layer neural network from Figure 3.

3. Construction of the Adjoint Neural Networks

In the previous section the possibility of conversion of a neural network into a flow graph has been shown, followed by properties of interreciprocal flow graphs (Krawczak 2002a). In this section these properties will allow us to introduce the *adjont neural networks*.

3.1. Adjoint networks

Using the graph theory notation a feedforward network topology can be specified by the following set of equations:

$$x_j = \sum_{i=1}^{N} T_{ij}\, w_{ij}\, x_i, \quad i, j = 1,2,\dots,N \tag{4}$$

where N is the number of all nodes (in our case neurons), x_i, x_j are values describing nodes i and j, $T_{ij} = f_{ij}(\ )$ is a transmittance (or an activation function of neurons, e.g. a sigmoid function, between the nodes (i and j). Summation is extended over all signals associated with the node x_i, $i = 1,2,\dots,N$, coming into the node x_j, $j = 1,2,\dots,N$.

The last equation can be rewritten in a different way by considering the values of nodes, i.e. $net_{j(l)}$, for $l = 1,2,\dots,L,\quad j(l) = 1,2,\dots,N(l)$, and has one of the form:

$$net_{j(l)} = \begin{cases} net_{i(0)} = x_{i(0)}, \; l = 0 \\ \sum\limits_{i(0)=1}^{N(0)} w_{i(0)j(1)}\, x_{i(0)}, \; l = 1 \\ \sum\limits_{i(l-1)=1}^{N(l-1)} w_{i(l-1)j(1)}\, f\left(net_{i(l-1)}\right), \; 1 < l < L \\ net_{j(out)} = \sum\limits_{i(L)=1}^{N(L)} w_{i(L-1j(L)}\, f\left(net_{i(L-1)}\right), \; l = L. \end{cases} \tag{5}$$

Using (5) we can illustrate flows of signals in a neural network treated as a flow graph in the following picture. An example is shown of a two-layer neural network with one hidden layer, Figure 5. A chain of directed edges linking a selected input node $net_{i(0)} = net_{i(in)}$ with a selected output node $net_{i(out)}$ is presented in the figure.

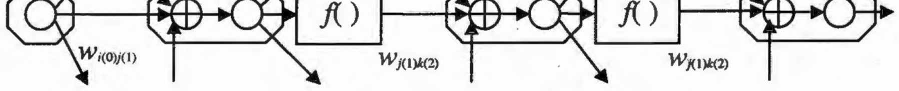

Fig.5. A schematic exemplary two-layer neural network as a flow graph.

Now let us recall the learning process of the neural networks. Learning of the neural networks consists in changing the weights when the desired output d_p, $p = 1,2,\ldots,P$, and the actual output $x_{(L)p}$, resulting both from the input $x_{(0)p}$, are different. The index p is a training example, while L denotes the number of layers in the network. The change is done by the gradient descent,

$$\Delta w_{i(l-1)j(l)} = w^{new}_{i(l-1)j(l)} - w^{old}_{i(l-1)j(l)} = -\eta \frac{\partial E}{\partial w_{i(l-1)j(l)}}, \tag{6}$$

where η is the learning rate, and E is the learning performance. Generally, the learning performance E is defined as the sum over all the training pattern examples:

$$E = \sum_{p=1}^{P} \sum_{j(L)=1}^{N(L)} E_{j(L)p}\left(d_{j(L)p}, x_{j(L)p}\right) \tag{7}$$

where $N(L)$ is a number of output nodes (neurons). For a specific training pattern p we use the squared difference between the patterns and the actual network output:

$$E_p = \frac{1}{2} \sum_{j(L)=1}^{N(L)} \left(d_{j(L)p} - x_{j(L)p}\right)^2 . \tag{8}$$

Using directly the delta rule for updating the weights $w_{i(l-1)j(l)}$, we obtain:

$$\Delta w_{i(l-1)j(l)} = -\eta \frac{\partial E_p}{\partial w_{i(l-1)j(l)}} = -\eta \frac{\partial E_p}{\partial net_{j(l)}} \frac{\partial net_{j(l)}}{\partial w_{i(l-1)j(l)}} = \eta \delta_{j(l)} x_{i(l-1)} \tag{9}$$

where $j(l) = 1,2,\ldots, N(l)$, $\quad i(l-1) = 1,2,\ldots, N(l-1)$, $\quad l = 1,2,\ldots, L$.

Let us recall the expression for δ in (9):

$$\delta_{i(l-1)} = \begin{cases} \delta_{j(out)} = \left(d_{j(L)p} - x_{j(L)p}\right) = e_j, & \text{for } l = 0 \\ \sum\limits_{j(L)=1}^{N(L)} f'\left(net_{j(L)}\right)\delta_{j(out)}, & \text{for } l = L \\ f'\left(net_{i(l-1)}\right) \sum\limits_{j(l)=1}^{N(l)} w_{i(l-1)j(l)}\, \delta_{j(l)}, & \text{for } 2 \le l \le L-1 \\ \delta_{i(0)} = \delta_{i(in)} = \sum\limits_{j(0)=1}^{N(0)} w_{i(0)j(1)}\, \delta_{j(1)}, & \text{for } l = 1 \end{cases} \tag{10}$$

It can be easily noticed that Equations 5 and 10 have the same structure. In (5) the signals *net* flow from the inputs through the network to the outputs, while in (10) the signals δ flow in the opposite direction, from the outputs to the inputs. In (10) the influence of a performance index of learning is included.

According to the definition of interreciprocity of flow graphs it is required to define the input to the adjoint graph. The term $\delta_{j(out)} = \left(d_{j(L)p} - x_{j(L)p}\right) = e_j$ depends on the shape of the performance index, and can be treated as the input to the adjoint network. In the original networks, the input nodes are distinguished, and the response of the network is simply $\delta_{j(out)} = e_j$. In order to demonstrate some similarity we must consider input nodes of the original network as output nodes of the adjoint network, and vice versa - the output nodes of the original network as input nodes of the adjoint network, equal to e_j.

Now we must determine the form of the transmittances of the reciprocal graph, i.e.:

$$T_{ij}^{T} = T_{ji}, \text{ for all } i, j = 1,2,\ldots,N. \tag{11}$$

For the normal signals flow direction we can write the following relationship:

$$net_{j(l)} = T_{i(l-1)j(l)}\, w_{i(l-1)j(l)}\, net_{i(l-1)} = f(net_{i(l-1)})\, w_{i(l-1)j(l)}\, net_{i(l-1)} \tag{12}$$

while for the opposite signal flow direction we can write:

$$\delta_{i(l-1)} = S_{j(l)i(l-1)}\, w_{i(l-1)j(l)}\, \delta_{j(l)} \tag{13}$$

where $S_{j(l)i(l-1)}$ is the transmittance for the delta signals. Let us differentiate (12) and rewrite (13) using the definition of the delta rule. We obtain:

$$\frac{\partial net_{j(l)}}{\partial net_{i(l-1)}} = \frac{\partial f(net_{i(l-1)})}{\partial net_{i(l-1}} w_{i(l-1)j(l)}, \tag{14}$$

$$\begin{gathered} \frac{\partial E}{\partial net_{i(l-1)}} = S_{j(l)i(l-1)}\ w_{i(l-1)j(l)} \frac{\partial E}{\partial net_{j(l)}}, \\ \frac{\partial net_{j(l)}}{\partial net_{i(l-1)}} = S_{j(l)i(l-1)}\ w_{i(l-1)j(l)} \frac{\partial E}{\partial E}, \end{gathered} \tag{15}$$

and by substituting (14) into (15) we get:

$$S_{j(l)i(l-1)} = T^T_{j(l)i(l-1)} = \frac{\partial f(net_i)}{\partial net_i} = f'(net_i). \tag{16}$$

In this way we have shown the form of the transposed transmittances appearing in the adjoint neural networks. In Figure 5 we have presented an exemplary path of a two-layer neural network, while a counterpart to this example - a path of the adjoint neural network is shown in Figure 6.

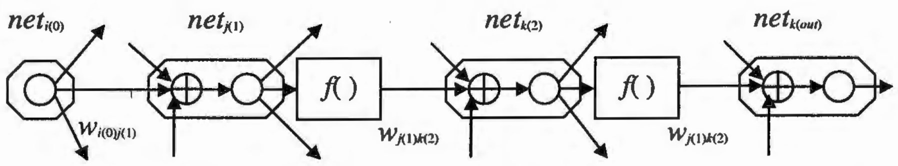

Fig.6. A schematic exemplary two-layer neural network as an adjoint neural network.

Looking at Equations 5 and 10 we can notice that both networks are topologically identical, meaning that there is a strict and unique correspondence between respective signals and connections. The adjoint network is found by application of the same network architecture, reversal of the direction of signal flows, replacement of activation functions by their derivatives, and switching of the positions of summing points with junction points within each node.

It is easy to notice that the transformation of the original network into the adjoint network is governed by very simple rules that will be described in the next section.

In the transposed graph the signals flow in the reverse direction to that of the original graph G and therefore the transposed graph becomes the *adjoint network*. The adjoint network is characterized by the inputs (which are the outputs of the original network) and the outputs (which are the inputs to the original network).

3.2. Neural networks versus networks

In some works of the present author, e.g (Krawczak 2000), the multilayer neural networks were divided into two kinds of layers. The first kind consisted only of weights with summation points while the second only of activation functions. The idea was just the same as in this paper, the simple weight layers being related to nodes and the activation function layers being related to edges with transmittances.

When considering the architecture of any class of neural networks, e.g. feedforward networks, we can observe four basic building elements of networks:

- summing points,
- junction points,
- univariate functions,
- multivariate functions (Krawczak 2000).

Any architecture of neural networks can be represented as a flow graph by introduction of the following changes:

- a neuron is divided into three parts: a summation point, a junction point and an activation function,
- a summation point together with a junction point becomes a node of a flow graph (of a network),
- an activation function becomes a transmittance of each outgoing edge,
- a transmittance is multiplied by a synaptic weight.

In the previous section we have shown the idea of changing a neural network into a flow network, and next deriving the adjoint network. Construction of an adjoint network requires the reversal of the flow direction in the original network, the labeling of all resulting signals as δ_j, and performing of the operations described above.

By reversing the signal flow, output nodes y_{out} in the considered network become input nodes in the adjoint network. These inputs are then taken to be ej. For cost functions different than the squared error considered here, the input should be set to $\partial E / \partial y_{out}$.

The rules considered allow for a simple construction of the adjoint network from the original network. Note that there is a topological equivalence between the two networks. The order of computations in the adjoint network is thus identical to the order of computations in the original network. The signals δ_j that propagate through the adjoint network correspond to the terms $\partial E/\partial x_j$ necessary for gradient adaptation. The exact equations can be taken from the adjoint network, which would complete the derivation.

4. Derivation of the Backpropagation Algorithm

The simplicity of application of the adjoint neural networks in deriving the error backpropagation algorithm will be clarified in this section.

We derive the standard backpropagation algorithm. For the sake of consistency with the traditional notation, we have labeled the summation signal $net_{i(L)}$ with the capital subscript denoting the layer. The adjoint network shown in Figure 6 is found by applying the construction rules described above. From this figure we may immediately write down the equations for calculating the delta terms, namely Equation 10.

Equation 9 for the weight update now can be formulated as

$$\Delta w_{i(l-1)j(l)} = \eta \delta_{j(l)} x_{i(l-1)} .$$

These two equations, (5) and (9), precisely describe the standard backpropagation, the equations well known from many textbooks about feedforward neural networks. There is no doubt that the presented approach gives a much more easy way to obtain the equations describing the backpropagation algorithm.

In this paper we have demonstrated the way for the derivation of the gradient-based algorithms for any network architectures. For any interesting neural network we can construct the adjoint neural network. The adjoint network is built using three rules of changing the original network, and can be obtained in a very simple way. All formulae for the learning algorithms can be derived using this methodology.

References

Bryson A. E., Ho Y-C. (1969) *Applied Optimal Control.* Blaisdell, Waltham, MA.

Chua L. O. and Lin P-M. (1975) *Computer-Aided Analysis of Electric Circuits*, Prentice-Hall, New Jersey.

Dreyfus S.E. (1990) Artificial Neural Networks, Back Propagation, and the Kelley-Bryson Gradient Procedure. *Journal of Guidance*, **13**, 5, 926-928.

Fettweiss A. (1972) A general theorem for signal-flow networks, with applications. in :*Digital Signal Processing,* IEEE Press, 126-130.

Hassoun M. H. (1995). *Fundamentals of Artificial Neural Networks* (MIT Press).

Hornik K., Stinchcombe M. and White H. (1989) *Multilayer feedforward networks are universal approximators*, Neural Networks, **2**, 359-366.

Krawczak M. and Mizukami K. (1994) The control theory approach to perceptron learning process. *44 Conference of IEE of Japan*, Okayama.

Krawczak M. (1999) Dynamic learning for feedforward neural networks. In: *Proceedings of the conference: ICONIP'99 ANZIIS'99&ANNES'99 &ACNN'99*, Perth, Western Australia, 16-20 November, 33-36.

Krawczak M. (2000) Backpropagation versus dynamic programming approach. *Bulletin of Polish Academy of Sciences.* **48**, 2, 167-180.

Krawczak M. (2001) Neural networks learning as a particular optimal control problem. *Proc. 7th Int. Conference on Methods and Models in Automation and Robotics*, IEEE, Międzyzdroje, Poland, August 2001.

Krawczak M. (2002) *Generalized Nets Representation of Neural Networks Learning.* Polish Academy of Sciences, Systems Research (in print).

Rall B. (1981). *Automatic Differentiation: Techniques and Applications,* Lecture Notes in Computer Science, Springer-Verlag.

Rumelhart D. E., Hilton G. E., Williams R. J. (1986*) Parallel Distributed Processing*. vol. 1, edited by D. Rumelhart, J. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, Chap. **8**.

Wan E., and Beaufays F. (1996) Diagrammatic derivation of gradient algorithms for neural networks. *Neural Computation,* **8**, 1, January, 182-206.

Werbos P. J. (1989). Maximizing Long-Term Gas Industry Profits in Two Minutes in Lotus Using Neural Network Methods. *IEEE Trans. on Systems, Man, and Cybernetics*, **19**, 2, 315-333.

Zurada J. M. (1992) *Introduction to artificial neural systems*, West Publishing Company, St. Paul.

ISBN 83-85847-78-2