Pruning of recurrent neural models: an optimal brain damage approach

Patryk Chaber · Maciej Ławryńczuk

Received: 13 March 2017 / Accepted: 18 January 2018 / Published online: 31 January 2018
© The Author(s) 2018. This article is an open access publication

Abstract This paper considers the problem of pruning recurrent neural models of perceptron type with one hidden layer which may be used for modelling of dynamic system. In order to reduce the number of model parameters (i.e. the number of weights), the Optimal Brain Damage (OBD) pruning algorithm is adopted for the recurrent neural models. Efficiency of the OBD algorithm is demonstrated for pruning neural models of a neutralisation reactor benchmark process. For the considered neutralisation system, the OBD algorithm makes it possible to reduce as many as 60% of model parameters and reduce the validation error by some 30% when compared to the full (unpruned) models.

Keywords Neural networks · Dynamic systems · Model pruning · Model structure optimisation

1 Introduction

Advanced control methods [1,2] as well as fault detection techniques [3,4] and fault-tolerant control approaches [5] directly use models of dynamic sys-

tems in order to make online decisions. Additionally, dynamic models are used in state estimation [6], simulation [7,8], time-series forecasting [9], numerical optimisation [10,11] and they are necessary for development of soft sensors [12]. Models are also necessary in recognition and interpretation of medical images [13]. That is why finding precise and uncomplicated models is the first step, but a fundamental one in development of the mentioned algorithms. In the mentioned advanced algorithms the model is used not only offline, during its development, but also for online calculations. For example, in Model Predictive Control (MPC) [2] an optimisation procedure calculates online at each sampling instant the best possible control policy considering future predictions of the dynamic model. A precise model results in excellent performance of the MPC controller, but, the opposite is also true, i.e. when the accuracy of the model is poor, the controller makes decisions using false predictions and the resulting control quality may be below expectations.

Two approaches may be used to find the model: modelling and identification. In the first case, all the phenomena taking place in the process must be described analytically which leads to a fundamental (first-principles) model [7,8]. Theoretically, the fundamental models have very good accuracy, but from a practical point of view they need many technological parameters, whose values may be difficult to determine. Moreover, in practice dynamic fundamental models may consist of many differential equations, whose online solution may be difficult and time-consuming

P. Chaber (✉) · M. Ławryńczuk
Institute of Control and Computation Engineering, Faculty
of Electronics and Information Technology, Warsaw
University of Technology, Warsaw, Poland
e-mail: P.Chaber@stud.elka.pw.edu.pl

M. Ławryńczuk
e-mail: M.Lawrynczuk@ia.pw.edu.pl

in predictive control, fault detection and fault-tolerant control. That is why black-box models are frequently used in many applications. In such cases, the structure of the model is chosen arbitrarily and its parameters are optimised in such a way that the discrepancy between model output and a recorded set of data is minimised [14]. Taking into account that a good model should be not only precise, but also that it is desirable to have a model which may be easily used in the aforementioned algorithms [15], one may easily conclude that neural networks of different structures [16, 17] are very good options. In particular, the recurrent neural models of perceptron type with one hidden layer [16, 18] are successfully used for approximation of numerous dynamic systems, e.g. a polystyrene batch chemical reactor [19], an ethylene-ethane distillation column and a polymerisation reactor [1], a neutralisation reactor [20], a fluid catalytic cracking unit [21].

Unlike fundamental models, the neural ones have a very simple structure and they do not consist of differential as well as algebraic equations, which greatly simplifies their usage. On the other hand, the basic question is the number of hidden nodes, which affects the overall number of model parameters (weights). The higher number of model parameters, the better accuracy for the training data set, but, at the same time, the higher risk of low generalisation ability. This means that too complex neural models tend to approximate specific data sets rather than to mimic behaviour of the dynamic processes. A frequent approach used in practice is to train neural models and next to remove the weights of the lowest importance (the process of pruning). As a result of pruning, one obtains networks of good accuracy and good generalisation, which also have a low number of parameters. There are numerous pruning methods, e.g. the Tukey–Kramer multiple comparison procedure [22], pruning using cross-validation [23], the pruning method optimised with a Particle Swarm Optimisation (PSO) algorithm [24], Bayesian regularisation [25], pruning using Minimum Validation error regulariser [26], Optimal Brain Damage (OBD) [27], optimal brain surgeon [28], and other novel approaches [29]. In particular, the OBD algorithm is very effective as reported in the literature. The strategy used in this algorithm assumes deleting parameters (setting its value permanently to 0) which have the least effect on the training error of the model. It has been successfully used to prune models next used in different fields, e.g. for monitoring of exhaust valves [30], in classification

of spectral features for automatic modulation recognition [31], in modelling two-mass drive system [25], in motor fault diagnosis [32], for load forecasting of a power system [33], for simultaneous determination of phenol isomers in binary mixtures [34], for microbial growth prediction in food [35]. The applications of the OBD algorithm reported in the literature are concerned with non-recurrent model configuration whereas in the case of dynamic systems the recurrent training mode is a straightforward option.

The motivation of this work is the necessity to obtain precise dynamic models capable of long-range prediction that have moderate number of parameters. In general, two model configurations are possible: serial-parallel and parallel [36]. In the non-recurrent serial-parallel one the model output signal is a function of the process input and output signal values from previous discrete sampling instants (real measurements). Hence, the serial-parallel model should be only used for one-step-ahead prediction. In the recurrent parallel configuration, the model output signal depends on its values at some previous sampling instants. Since in MPC, fault detection, fault-tolerant control, process optimisation and simulation it is necessary to calculate precise predictions of the process output variable over long horizons (for multiple steps ahead), it is obvious that for such applications the recurrent parallel model should be used, not the simple non-recurrent one. In the context of MPC, demonstration of this fact for linear models is discussed in [37, 38], considerations for nonlinear models are given in [39, 40]. Finally, it is essential to stress the fact that the models characterised by a moderate number of parameters are preferred. It is important not only because such models have good generalisation ability, but also because in practical applications resources of computational units used in online process control, fault detection and optimisation are typically limited and models with too many parameters are likely to slow down calculations repeated in real time.

Contribution of this work is twofold. Firstly, the rudimentary OBD pruning algorithm [27] is derived for a particular model of recurrent dynamic model—a neural network with one hidden layer. Implementation details of the algorithm are given. Since the focus is entirely on the recurrent neural network, this work is an extension of the original paper [27] in which the OBD algorithm has been introduced, but only static models have been considered. Secondly, effectiveness of the derived OBD algorithm for recurrent neural

models is demonstrated for a neutralisation (pH) reactor which is a classical benchmark in process control. The process has significantly nonlinear steady-state and dynamic properties and is frequently used to compare dynamic models and their identification algorithms as well as advanced control methods (e.g. [41–46]). Detailed discussion how to use the described algorithm to obtain precise models with good generalisation ability is given. The remainder of the paper is organised in the following way. Firstly, in Sect. 2, the structure of the neural model is defined and its training algorithm is shortly discussed. Section 3, which is the main part of the paper, details the OBD algorithm for the recurrent neural models. Next, Sect. 4 presents simulation results concerned with training and pruning of recurrent neural models of a neutralisation process. Finally, Sect. 5 concludes the paper.

2 Neural dynamic model

2.1 Structure of the model

It is assumed that the input variable of the dynamic process under consideration is denoted by u and the output variable is denoted by y . There are two possible configurations of dynamic models: serial-parallel and parallel [36]. In the non-recurrent serial-parallel structure the output signal of the model for the current sampling instant k , $y^{\text{mod}}(k)$, is a function $f: \mathbb{R}^{n_A+n_B-\tau+1} \rightarrow \mathbb{R}$ of the process input and output signal values from some previous instants

$$y^{\text{mod}}(k) = f(u(k-\tau), \dots, u(k-n_B), \\ y(k-1), \dots, y(k-n_A)),$$

where the positive integers n_A and n_B define the order of model dynamics and $\tau \leq n_B$ is the time-delay. In the recurrent parallel model, which may be also named the simulation model, the past process outputs are replaced by the model outputs calculated at previous sampling instants, i.e. the output signal of the model is a function of the past process input signal values and of the output signal values calculated from the model at some previous instants

$$y^{\text{mod}}(k) = f(u(k-\tau), \dots, u(k-n_B), \\ y^{\text{mod}}(k-1), \dots, y^{\text{mod}}(k-n_A)). \quad (1)$$

The serial-parallel structure is a one-step-ahead predictor since the real measurements of the process output variable are necessary whereas the parallel structure is more appropriate not only for simulation, but also in applications where the very recent measurements are not available, e.g. in long-range prediction, including MPC algorithms, model-based fault detection and fault isolation.

As the dynamic model of the process the neural network the structure of which is depicted in Fig. 1 is used. The most popular Multi-Layer Perceptron (MLP) feedforward neural network structure with two layers is used [16, 17]. The network has $n_A + n_B - \tau + 1$ input nodes, which corresponds with the arguments of the general parallel model (1), K nonlinear hidden nodes with the nonlinear transfer function $\varphi: \mathbb{R} \rightarrow \mathbb{R}$, e.g. of the tanh type, one linear output element (sum) and one output $y^{\text{mod}}(k)$. The additional constant unitary inputs of the first and the second layers are biases. The weights of the first layer are denoted by $w_{i,j}^1$, where $i = 1, \dots, K$, $j = 0, \dots, n_A + n_B - \tau + 1$, the weights of the second layer are denoted by w_i^2 , where $i = 0, \dots, K$. The output signal of the neural network can be expressed by

$$y^{\text{mod}}(k) = w_0^2 + \sum_{i=1}^K w_i^2 v_i(k) = w_0^2 + \sum_{i=1}^K w_i^2 \varphi(z_i(k)), \quad (2)$$

where the output signal of the i th hidden node is denoted by $v_i(k)$, $z_i(k)$ is the sum of input signals of the i th hidden node:

$$z_i(k) = w_{i,0}^1 + \sum_{j=1}^{I_u} w_{i,j}^1 u(k-\tau+1-j) \\ + \sum_{j=1}^{n_A} w_{i,I_u+j}^1 y^{\text{mod}}(k-j), \quad (3)$$

and $I_u = n_B - \tau + 1$. From Eqs. (2) and (3), the output signal of the network is

$$y^{\text{mod}}(k) = w_0^2 + \sum_{i=1}^K w_i^2 \varphi \left(w_{i,0}^1 + \sum_{j=1}^{I_u} w_{i,j}^1 u(k-\tau+1-j) \right. \\ \left. + \sum_{j=1}^{n_A} w_{i,I_u+j}^1 y^{\text{mod}}(k-j) \right). \quad (4)$$

Fig. 1 Structure of the neural model

2.2 Training of the model

The objective of training is to find such a set of weights so that the value of the model error has an acceptable minimal value. The error of the model is defined by the following sum of squared errors

$$E(\mathbf{w}) = \sum_{k=S}^P \left(y^{\text{mod}}(k) - y(k) \right)^2, \quad (5)$$

where $y^{\text{mod}}(k)$ is the output of the neural network for the sampling instant k , $y(k)$ is an output signal of the real process (the training pattern), $S = \max\{n_A, n_B\} + 1$, P is the number of training samples and all the weights form a vector of parameters

$$\mathbf{w} = \left[w_{1,0}^1 \dots w_{1,n_A+n_B-\tau+1}^1 \dots w_{K,0}^1 \dots w_{K,n_A+n_B-\tau+1}^2 \dots w_K^2 \right]^T.$$

During training, the model error function (5) is minimised. Due to the nonlinear hidden layer transfer function φ , this is an unconstrained nonlinear optimisation problem. The general gradient-based training algorithm, leading to minimisation of the model error function (5) may be summarised in the following steps (the consecutive iterations are denoted by $t = 1, \dots, t^{\text{max}}$) [47–49]:

0. Initialisation of the weights $\mathbf{w}$, random values are usually chosen from the range $-1, 1 >$.
1. The model output signal $y^{\text{mod}}(k)$ for the sampling instants $k = S, \dots, P$ and for the current weights $\mathbf{w}$ are calculated from Eq. (4).
2. The model error for the whole training data set is calculated from Eq. (5).

3. If the model error or a norm of its gradient satisfies a stopping criterion, the algorithm is stopped.
4. The optimisation direction $\mathbf{p}_t$ is calculated.
5. The optimal step-length η_t along the direction $\mathbf{p}_t$ is calculated using, e.g. the golden section approach or the Armijo's rule [48].
6. The model weights are updated $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \mathbf{p}_t$, the training algorithm goes to step 1.

The following stopping criteria may be used:

- the training algorithm terminates when the maximal number of iterations is exceeded, i.e. when $t > t_{\text{max}}$,
- the algorithm terminates when the change of weights in two consecutive iterations is smaller than arbitrarily small quantity $\varepsilon_{\Delta \mathbf{w}} > 0$, i.e. when $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| < \varepsilon_{\Delta \mathbf{w}}$,
- the algorithm terminates when the norm of the gradient of the minimised error function is small, i.e. when $\left\| \frac{dE(\mathbf{w})}{d\mathbf{w}} \right\|_{\mathbf{w}=\mathbf{w}_t} < \varepsilon_{\nabla E}$, where $\varepsilon_{\nabla E}$ is an arbitrarily small positive quantity.

The simplest approach to find the optimisation direction is to use the steepest-descent technique, in which the direction is opposite to the current gradient of the model error with respect to the optimised weights [47–49], i.e.

$$\mathbf{p}_t = - \left. \frac{dE(\mathbf{w})}{d\mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t},$$

where

$$\left. \frac{dE(\mathbf{w})}{d\mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t} = \left[\left. \frac{\partial E(\mathbf{w})}{\partial w_{1,0}^1} \right|_{\mathbf{w}=\mathbf{w}_t} \dots \left. \frac{\partial E(\mathbf{w})}{\partial w_{K,I}^1} \right|_{\mathbf{w}=\mathbf{w}_t} \dots \left. \frac{\partial E(\mathbf{w})}{\partial w_0^2} \right|_{\mathbf{w}=\mathbf{w}_t} \dots \left. \frac{\partial E(\mathbf{w})}{\partial w_K^2} \right|_{\mathbf{w}=\mathbf{w}_t} \right]^T,$$

and $I = n_A + n_B - \tau + 1$. Due to very slow convergence of the steepest-descent method, a quasi-Newton algorithms [47–49] are recommended in this work. In these algorithms, the direction is calculated from the general formula given by

$$\mathbf{p}_t = - [H(\mathbf{w}_t)]^{-1} \left. \frac{dE(\mathbf{w})}{d\mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t},$$

$$\mathbf{H}(\mathbf{w}_t) = \frac{d^2 E(\mathbf{w})}{d\mathbf{w}^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial (w_{1,0}^1)^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_{1,0}^1 \partial w_{K,I}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \frac{\partial E(\mathbf{w})}{\partial w_{1,0}^1 \partial w_0^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_{1,0}^1 \partial w_K^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_{K,I}^1 \partial w_{1,0}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial (w_{K,I}^1)^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \frac{\partial E(\mathbf{w})}{\partial w_{K,I}^1 \partial w_0^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_{K,I}^1 \partial w_K^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} \\ \frac{\partial E(\mathbf{w})}{\partial w_0^2 \partial w_{1,0}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_0^2 \partial w_{K,I}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \frac{\partial E(\mathbf{w})}{\partial (w_0^2)^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_0^2 \partial w_K^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_K^2 \partial w_{1,0}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial w_K^2 \partial w_{K,I}^1} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \frac{\partial E(\mathbf{w})}{\partial w_K^2 \partial w_0^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} & \cdots & \frac{\partial E(\mathbf{w})}{\partial (w_K^2)^2} \bigg|_{\mathbf{w}=\mathbf{w}_t} \end{bmatrix} \quad (6)$$

where $\mathbf{H}(\mathbf{w}_t)$ is the Hessian matrix of the error function $E(\mathbf{w}_t)$, the structure of which is given by Eq. (6). Because analytical calculation of the inverse of the Hessian matrix, i.e. $[\mathbf{H}(\mathbf{w}_t)]^{-1}$, is quite complex, it is not calculated analytically, but approximated numerically. In this work, a very efficient Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is used [47–49]. In each iteration of the training (weight optimisation) algorithm the inverse Hessian $[\mathbf{H}(\mathbf{w}_t)]^{-1}$ is approximated by the matrix $\mathbf{V}_t$ from the formula

$$\mathbf{V}_t = \mathbf{V}_{t-1} + \left[1 + \frac{\mathbf{r}_t^T \mathbf{V}_{t-1} \mathbf{r}_t}{s_t^T \mathbf{r}_t} \right] \frac{s_t s_t^T}{s_t^T \mathbf{r}_t} - \frac{s_t \mathbf{r}_t^T \mathbf{V}_{t-1} + \mathbf{V}_{t-1} \mathbf{r}_t s_t^T}{s_t^T \mathbf{r}_t},$$

where the increment of the weights vector is $s_t = \mathbf{w}_t - \mathbf{w}_{t-1}$, increment of the gradient vector of the weights vector is denoted by $\mathbf{r}_t = \frac{dE(\mathbf{w}_t)}{d\mathbf{w}_t} - \frac{dE(\mathbf{w}_{t-1})}{d\mathbf{w}_{t-1}}$. The gradients of the error function are determined analytically at each iteration of the training algorithm. Differentiating Eq. (5) with respect to the weights of the first and the second layer, one obtains

$$\frac{dE(\mathbf{w})}{dw_{i,j}^1} = 2 \sum_{k=S}^P \left(y^{\text{mod}}(k) - y(k) \right) \frac{\partial y^{\text{mod}}(k)}{\partial w_{i,j}^1}, \quad (7)$$

for all $i = 1 \dots K, j = 0, \dots, n_A + n_B - \tau + 1$ and

$$\frac{dE(\mathbf{w})}{dw_i^2} = 2 \sum_{k=S}^P \left(y^{\text{mod}}(k) - y(k) \right) \frac{\partial y^{\text{mod}}(k)}{\partial w_i^2}, \quad (8)$$

for all $i = 1, \dots, K$. Next, differentiating Eq. (2), one has

$$\begin{aligned} \frac{\partial y^{\text{mod}}(k)}{\partial w_{i,j}^1} &= \sum_{n=1}^K w_n^2 \frac{\partial v_n(k)}{\partial w_{i,j}^1} \\ \frac{\partial y^{\text{mod}}(k)}{\partial w_i^2} &= \sum_{n=1}^K w_n^2 \frac{\partial v_n(k)}{\partial w_i^2} + v_i(k), \end{aligned} \quad (9)$$

where

$$v_n(k) = \begin{cases} 1 & \text{for } n = 0 \\ \varphi(z_n(k)) & \text{for } n \neq 0 \end{cases}.$$

Taking into account that $v_i(k) = \varphi(z_i(k))$ (Eq. 2)

$$\frac{\partial v_n(k)}{\partial w_{i,j}^1} = \frac{\partial \varphi(z_n(k))}{\partial z_n(k)} \frac{\partial z_n(k)}{\partial w_{i,j}^1}, \quad \frac{\partial v_n(k)}{\partial w_i^2} = \frac{\partial \varphi(z_n(k))}{\partial z_n(k)} \frac{\partial z_n(k)}{\partial w_i^2}. \quad (10)$$

where the partial derivative $\frac{\partial \varphi(z_n(k))}{\partial z_n(k)}$ depends on the type of the transfer function used. When $\varphi(z_n(k)) = \tanh(z_n(k))$

$$\frac{\partial \varphi(z_n(k))}{\partial z_n(k)} = 1 - \tanh^2(z_n(k)) \quad (11)$$

Differentiating Eq. (3) with respect to the weights of the first layer gives

$$\frac{\partial z_n(k)}{\partial w_{i,j}^1} = \begin{cases} x_j(k) + \sum_{k_0=1}^{n_A} w_{n,I_u+k_0}^1 \frac{\partial y^{\text{mod}}(k-k_0)}{\partial w_{i,j}^1} & \text{for } n = i \\ \sum_{k_0=1}^{n_A} w_{n,I_u+k_0}^1 \frac{\partial y^{\text{mod}}(k-k_0)}{\partial w_{i,j}^1} & \text{for } n \neq i \end{cases}, \quad (12)$$

where $x_0(k) = 1$, $x_i(k) = u(k - \tau + 1 - j)$ for $i = 1, \dots, I_u$, $x_{I_u+i}(k) = y^{\text{mod}}(k - k_i)$ for $i = 1, \dots, n_A$. Differentiating Eq. (3) with respect to the weights of the second layer leads to

$$\frac{\partial z_n(k)}{\partial w_i^2} = \sum_{k_0=1}^{n_A} w_{n, I_u+k_0}^1 \frac{\partial y^{\text{mod}}(k - k_0)}{\partial w_i^2}. \quad (13)$$

The above formulae are universal for the considered recurrent neural model. When an alternative transfer function is used in place of the tanh function, it is only necessary to calculate the first-order derivatives $\frac{\partial \varphi(z_n(k))}{\partial (z_n(k))}$, $n = 1, \dots, K$, used in Eqs. (10) for the specific transfer function.

Two data sets are used: the training data set and the validation set. The first set is used only for model training, i.e. the value of the error function E is minimised only for this set. In order to assess generalisation ability of the trained model, the value of the error is also calculated for the validation set. Model selection, e.g. among a few compared models of different structures and/or initial weights, is accomplished taking into account only the validation error.

3 Pruning of the neural dynamic model

In the Optimal Brain Damage (OBD) pruning algorithm [27] the weights of the neural network with small saliency [i.e. those whose removal have the least influence on the error (5)] are deleted. In order to do so, a local model of the error function is formulated and the effect of perturbing the weights is analysed. The minimised error function (5) is approximated by means of a Taylor series. A perturbation $\Delta \mathbf{w}$ of the weight vector $\mathbf{w}$ changes the error function by

$$\begin{aligned} \Delta E &= \sum_i \Delta g_i \Delta w_i \\ &+ \frac{1}{2} \left(\sum_i h_{ii} (\Delta w_{ii})^2 + \sum_{i \neq j} h_{ij} \Delta w_i \Delta w_j \right) \\ &+ O(\|\Delta \mathbf{w}\|^3), \end{aligned} \quad (14)$$

where Δw_i denotes the perturbation of the i th weight, g_i is the i th element of the gradient vector with respect to the weight w_i , i.e. $g_i = \frac{\partial E}{\partial w_i}$, h_{ij} denotes the element of the Hessian matrix, i.e. the second-order derivative

of the error E , $h_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$. It is not recommended to remove weights during training because small saliency of the network with respect to some weight may result from a temporary value of that weight or because the value of the network error is huge. Therefore, it is recommended to remove weights after completing training. In the OBD algorithm, it is assumed that a local or a global minimum of the error function E is reached. In such a case, all gradients of the error with respect to weights are (approximately) 0 and the first term of the right side in Eq. (14) may be neglected. It is also assumed that the Hessian matrix $\mathbf{H}$ (6) is positive-definite and diagonally dominant. Hence, only the diagonal elements h_{ii} are considered, off-diagonal elements are assumed to be 0. Therefore, in the OBD algorithm the following quadratic approximation is used in place of Eq. (14)

$$\Delta E = \frac{1}{2} \sum_i h_{ii} (\Delta w_{ii})^2.$$

In the OBD algorithm a weight of the network is removed when its saliency is small. For the first layer the saliency is

$$S_{i,j}^1 = \frac{1}{2} \frac{\partial^2 E}{\partial (w_{i,j}^1)^2} (w_{i,j}^1)^2, \quad (15)$$

where $i = 1, \dots, K$, $j = 0, \dots, n_A + n_B - \tau + 1$ whereas for the second layer the saliency is

$$S_i^2 = \frac{1}{2} \frac{\partial^2 E}{\partial (w_i^2)^2} (w_i^2)^2, \quad (16)$$

where $i = 0, \dots, K$. To carry out the OBD procedure, the following steps must be considered:

1. The initial structure of the full network is selected and the network is trained (a local or a global minimum of the error function E is reached).
2. The second-order derivatives of the error function with respect to all the weights are calculated, i.e. $\frac{\partial^2 E}{\partial (w_{i,j}^1)^2}$ for $i = 1, \dots, K$, $j = 0, \dots, n_A + n_B - \tau + 1$ and $\frac{\partial^2 E}{\partial (w_i^2)^2}$ for $i = 0, \dots, K$.
3. The saliency value ($S_{i,j}^1$ and S_i^2) for each model weight is calculated using Eqs. (15) and (16).
4. The weights are sorted by their saliency value and then some weights with the lowest saliency value are deleted.
5. The pruned network is retrained (a local or a global minimum of the error function E is reached).

6. The algorithm returns to step 2.

The saliency value of each weight is calculated using the training data set. After the training, the validation error is calculated and assessed. If its value before and after removing one or some more weights increases too much, the OBD algorithm is stopped and the network before the last pruning is used.

Calculation of the second-order derivatives of error function, with respect to weights of the neural network, is done analytically. Later these values are used to determine saliency from Eqs. (15) and (16). Differentiating Eq. (7) gives

$$\frac{\partial^2 z_n(k)}{\partial (w_{i,j}^1)^2} = \begin{cases} 2 \frac{\partial y^{\text{mod}}(k-j+I_u)}{\partial w_{i,j}^1} + \sum_{k_0=1}^{n_A} w_{n,I_u+k_0}^1 \frac{\partial^2 y^{\text{mod}}(k-k_0)}{\partial (w_{i,j}^1)^2} & \text{for } n=i, j > I_u \\ \sum_{k_0=1}^{n_A} w_{n,I_u+k_0}^1 \frac{\partial^2 y^{\text{mod}}(k-k_0)}{\partial (w_{i,j}^1)^2} & \text{otherwise} \end{cases} \quad (20)$$

$$\frac{\partial^2 E(\mathbf{w})}{\partial (w_{i,j}^1)^2} = 2 \sum_{k=S}^P \left[\left(\frac{\partial y^{\text{mod}}(k)}{\partial w_{i,j}^1} \right)^2 + 2 (y^{\text{mod}}(k) - y(k)) \frac{\partial^2 y^{\text{mod}}(k)}{\partial (w_{i,j}^1)^2} \right],$$

for all $i = 1 \dots K, j = 0, \dots, n_A + n_B - \tau + 1$ whereas differentiating Eq. (8) results in

$$\frac{\partial^2 E(\mathbf{w})}{\partial (w_i^2)^2} = 2 \sum_{k=S}^P \left[\left(\frac{\partial y^{\text{mod}}(k)}{\partial w_i^2} \right)^2 + 2 (y^{\text{mod}}(k) - y(k)) \frac{\partial^2 y^{\text{mod}}(k)}{\partial (w_i^2)^2} \right],$$

for all $i = 1, \dots, K$. Next, differentiating Eqs. (9) one obtains

$$\frac{\partial^2 y^{\text{mod}}(k)}{\partial (w_{i,j}^1)^2} = w_i^2 \frac{\partial^2 v_i(k)}{\partial (w_{i,j}^1)^2}$$

and

$$\frac{\partial^2 y^{\text{mod}}(k)}{\partial (w_i^2)^2} = \begin{cases} 2 \frac{\partial v_i(k)}{\partial w_i^2} + w_i^2 \frac{\partial^2 v_i(k)}{\partial (w_i^2)^2} & \text{for } i \neq 0 \\ w_i^2 \frac{\partial^2 v_i(k)}{\partial (w_i^2)^2} & \text{for } i = 0 \end{cases}.$$

Differentiating Eqs. (10) gives

$$\frac{\partial^2 v_n(k)}{\partial (w_{i,j}^1)^2} = \frac{\partial^2 \varphi(z_n(k))}{\partial (z_n(k))^2} \left(\frac{\partial z_n(k)}{\partial w_{i,j}^1} \right)^2 + \frac{\partial \varphi(z_n(k))}{\partial z_n(k)} \frac{\partial^2 z_n(k)}{\partial (w_{i,j}^1)^2} \quad (17)$$

and for the second layer

$$\frac{\partial^2 v_n(k)}{\partial (w_i^2)^2} = \frac{\partial^2 \varphi(z_n(k))}{\partial (z_n(k))^2} \left(\frac{\partial z_n(k)}{\partial w_i^2} \right)^2 + \frac{\partial \varphi(z_n(k))}{\partial z_n(k)} \frac{\partial^2 z_n(k)}{\partial (w_i^2)^2}, \quad (18)$$

where the partial derivative $\frac{\partial^2 \varphi(z_n(k))}{\partial (z_n(k))^2}$ depends on the type of the transfer function used. When $\varphi(z_n(k)) = \tanh(z_n(k))$

$$\frac{\partial^2 \varphi(z_n(k))}{\partial (z_n(k))^2} = -2 \tanh(z_n(k))(1 - \tanh^2(z_n(k))) \quad (19)$$

Finally, differentiating Eq. (12) leads to Eq. (20)

whereas from Eq. (13) one obtains

$$\frac{\partial^2 z_n(k)}{\partial (w_i^2)^2} = \sum_{k_0=1}^{n_A} w_{n,I_u+k_0}^1 \frac{\partial^2 y^{\text{mod}}(k-k_0)}{\partial (w_i^2)^2}.$$

The above formulae are universal for the considered recurrent neural model. When an alternative transfer function is used in place of the tanh function, it is only necessary to calculate the second-order derivatives $\frac{\partial^2 \varphi(z_n(k))}{\partial (z_n(k))^2}$, $n = 1, \dots, K$, used in Eqs. (17) and (18) for the specific transfer function.

4 Simulation results

4.1 Process description

The process under consideration, shown schematically in Fig. 2, is a pH neutralisation reactor [50]. The reactor is a classical benchmark used for model identification and control, e.g. [20]. A base (NaOH) stream q_1 , a buffer (NaHCO₃) stream q_2 and an acid (HNO₃) stream q_3 are mixed in a constant volume tank. The output pH may be controlled by manipulating the base flow rate q_1 (ml/s). The buffer and acid streams are assumed to be constant ($q_2 = 0.55$ ml/s, $q_3 = 16.60$ ml/s). Therefore, the reactor has one input (manipulated) variable q_1 and one output (controlled) variable

Fig. 2 Schematic representation of the pH neutralisation process

pH. The reactor is described by two continuous-time nonlinear ordinary differential equations

$$\frac{dW_a(t)}{dt} = \frac{q_1(t)(W_{a1} - W_a(t))}{V} + \frac{q_2(t)(W_{a2} - W_a(t))}{V} + \frac{q_3(t)(W_{a3} - W_a(t))}{V}, \quad (21)$$

$$\frac{dW_b(t)}{dt} = \frac{q_1(t)(W_{b1} - W_b(t))}{V} + \frac{q_2(t)(W_{b2} - W_b(t))}{V} + \frac{q_3(t)(W_{b3} - W_b(t))}{V} \quad (22)$$

and one algebraic output equation

$$0 = W_a(t) + 10^{\text{pH}(t)-14} - 10^{-\text{pH}(t)} + W_b(t) \frac{1 + 2 \times 10^{\text{pH}(t)-\text{pK}_2}}{1 + 10^{\text{pK}_1-\text{pH}(t)} + 10^{\text{pH}(t)-\text{pK}_2}}. \quad (23)$$

State variables are reaction invariants: W_a is a charge-related quantity, W_b is the concentration of the carbonate ion. Parameters of the fundamental model defined by Eqs. (21), (22) and (23) are given in Table 1. The initial operating conditions are: $q_1 = 15.55$ ml/s, $q_2 = 0.55$ ml/s, $q_3 = 16.60$ ml/s, $\text{pH} = 7$, $W_a = -4.32 \times 10^{-4}$ mol, $W_b = 5.28 \times 10^{-4}$ mol.

4.2 Training of the initial full model

At first the fundamental model given by Eqs. (21), (22) and (23) is simulated for a random sequence of steps in the input variable, the sampling time is 10 s. As a result

Table 1 Parameters of the fundamental model of the pH neutralisation process

$W_{a1} = -3.05 \times 10^{-3}$ mol	$W_{b1} = 5 \times 10^{-5}$ mol	$V = 2900$ ml
$W_{a2} = -3 \times 10^{-2}$ mol	$W_{b2} = 3 \times 10^{-2}$ mol	$\text{pK}_1 = 6.35$
$W_{a3} = 3 \times 10^{-3}$ mol	$W_{b3} = 0$ mol	$\text{pK}_2 = 10.25$

Fig. 3 Input (q_1) and output (pH) process variables used for model training

Fig. 4 Input (q_1) and output (pH) process variables used for model validation

the training and validation data sets are obtained as shown in Figs. 3 and 4, respectively. The output signal in the data sets contains small measurement noise.

In order to eliminate the problem with saturation of hidden nodes, the process variables q_1 and pH are scaled

Fig. 5 The neural models with 20 hidden nodes: **a** the initial full networks N_{20}^1 , N_{20}^2 and N_{20}^3 , **b** the pruned network N_{20}^1 , **c** the pruned network N_{20}^2 , **d** the pruned network N_{20}^3

$$u = \frac{q_1 - q_{10}}{15}, \quad y = \frac{\text{pH} - \text{pH}_0}{5},$$

where in the initial operating point $q_{10} = 15.55$ ml/s, $\text{pH}_0 = 7$. Analogously to some previous work [51, 52], the second-order of model dynamics is used, i.e. $\tau = 1$, $n_A = n_B = 2$. From Eq. (1), the recurrent model is described by the following general relation

$$y^{\text{mod}}(k) = f(u(k-1), u(k-2), y^{\text{mod}}(k-1), y^{\text{mod}}(k-2)).$$

In this study two different configurations of the MLP neural model are considered: the network with 20 hidden nodes (Fig. 5a) and the network with 30 hidden nodes (Fig. 6a). The initial full models (i.e. with all weights) have quite a large number of parameters: the first structure has as many as 121 weights whereas the second one—181 weights. Because training is a non-linear optimisation problem of the model error function (5), which may be badly affected by local minima, for each model configuration as many as 10 networks with different initial weights initialised randomly are trained and pruned. The results presented next show the best 3 networks for each model configuration. All models are trained using the BFGS nonlinear optimisation algorithm, the golden section procedure is used

Fig. 6 The neural models with 30 hidden nodes: **a** the initial full networks N_{30}^1 , N_{30}^2 and N_{30}^3 , **b** the pruned network N_{30}^1 , **c** the pruned network N_{30}^2 , **d** the pruned network N_{30}^3

for step-length calculation. Because in the OBD algorithm it is assumed that before pruning the network is well trained, during training of the initial full models the maximal number of iterations of the BFGS algorithm is 2500 and the stopping criterion is defined as

$$\left\| \frac{dE(w)}{dw} \right\|_{w=w_t} \leq 10^{-9} \text{ or } \|w_t - w_{t-1}\| \leq 10^{-9}.$$

The chosen full neural models (trained) with 20 hidden nodes are denoted by N_{20}^1 , N_{20}^2 and N_{20}^3 , the full models with 30 nodes are denoted by N_{30}^1 , N_{30}^2 and N_{30}^3 . Training and validation errors for the full networks are given in Table 2. In general, the full networks are able to approximate behaviour of the process quite well, but it is interesting to note that although all networks have similar values of the training error, the validation error in some cases is significantly bigger. Such property of the networks may be associated with the fact that they have too many parameters as they tend to work fine only for the training data set. Figure 7 compares the validation data set and the outputs of the best initial full network with 20 hidden nodes—the network N_{20}^2 . Figure 8 depicts a similar comparison for the best network with 30 nodes, that is the structure N_{30}^1 . Inaccuracy of the models can be noticed especially for the samples 300–350 and 550–600. Figures 9 and 10 depict the difference between the output signal taken

Table 2 Training (E_t) and validation (E_v) errors for the initial full networks

Network	No. of weights	E_t	E_v
N_{20}^1	121	0.1785	1.0763
N_{20}^2	121	0.1907	0.8960
N_{20}^3	121	0.2094	1.2360
N_{30}^1	181	0.1672	0.5958
N_{30}^2	181	0.2505	3.2746
N_{30}^3	181	0.1839	1.1960

Fig. 7 Validation data set (solid line) versus the output of initial full network N_{20}^2 with 20 hidden nodes (dashed line) **Fig. 8** Validation data set (solid line) versus the output of initial full network N_{30}^1 with 30 hidden nodes (dashed line)

from the validation data set and the output signal calculated using the model with 20 hidden neurons and 30 hidden neurons, respectively.

Fig. 9 Error values for each sample k , calculated using validation data set and the output of initial full network: **a** N_{20}^1 , **b** N_{20}^2 , **c** N_{20}^3

4.3 Model pruning

Pruning of models is performed by deleting one weight at each iteration of the OBD algorithm. The number of iterations of the BFGS algorithm to retrain the model after weight removal varies from 2 to about 2000. Changes of model errors for the training and validation data sets are depicted in Figs. 11 and 12 for three initially considered networks with $K = 20$ and $K = 30$ hidden nodes, respectively. Additionally, Tables 3 and 4 give values of the training and validation errors for networks with removed selected numbers of weights. It is clear that deleting weights causes firstly drop and then the raise of both training and validation errors. This behaviour is expected, because each removal causes model to move from the point where the error function gradient is zeroed, and therefore escaping from a local minimum. What is more, there are less optimisation parameter, therefore there are less local minima of the error function which is minimised. Unfortunately removal of the weight might cause the model to be moved closer to local minimum which yields lower

Fig. 10 Error values for each sample k , calculated using validation data set and the output of initial full network: **a** N_{30}^1 , **b** N_{30}^2 , **c** N_{30}^3

model quality—it is visible as rapid increase in the error values. Global optimisation algorithms might be used to try to obtain more stable behaviour. When the model is oversimplified for the process to be approximated accurately, then the error values start to climb up. During experiments, it is observed that the classical stopping criterion of the OBD algorithm (i.e. termination of the algorithm if the validation error grows) may be misleading. For example, for the initial network N_{20}^1 the validation error grows rapidly after removing 28 weights (Fig. 11a), which may suggest that there is no point in continuing the OBD algorithm. Nevertheless, it is continued and although the next iteration gives an additional increase in the error, after removing 30 weights the validation error drops rapidly, resulting in even better fitted model than the original one. Similar significant temporal increases in the validation error may be observed after removing 22 and 55 weights

from the initial network N_{20}^2 (Fig. 11b) and after removing 45 weights from the network N_{20}^3 (Fig. 11c). That is why it is reasonable to continue to remove weights even if the validation error indicates that the

Fig. 11 Training (E_t , dashed line) and validation (E_v , solid line) errors for the networks with $K = 20$ hidden nodes after removing the given number of weights in consecutive iterations of the OBD algorithm: **a** pruning of the initial full network N_{20}^1 , **b** pruning of the initial full network N_{20}^2 , **c** pruning of the initial full network N_{20}^3

whole pruning procedure should be stopped. Finally, when too many weights are removed from the network, the model error grows permanently which means that further pruning must be stopped.

Table 5 includes training and validation error values calculated for fully pruned models (when there are no more weights that can be pruned and the model cannot be trained any further) and the number of resulting weights. As it was mentioned, although the number of parameters is low, the error values are unacceptable. It is a question how to choose a satisfying compromise between model accuracy and its simplicity. Taking that into consideration, only the models that are considered to have reasonable accuracy and a relatively low number of parameters are chosen. Training and validation errors of such models and their number of weights are given in Table 6. Their errors are significantly lower when compared to fully pruned models, and the numbers of weights left are maintained on the

Fig. 12 Training (E_t , dashed line) and validation (E_v , solid line) errors for the networks with $K = 30$ hidden nodes after removing the given number of weights in consecutive iterations of the OBD algorithm: **a** pruning of the initial full network N_{30}^1 , **b** pruning of the initial full network N_{30}^2 , **c** pruning of the initial full network N_{30}^3

similar level. Furthermore, their errors are quite similar to the errors of the full, unpruned models (Table 2). Figures 13 and 14 depict the difference between data

set and model output signal values for models with 20 hidden neurons and 30 hidden neurons, respectively. Taking into account Tables 2 and 6, one may easily find changes of the number of parameters and model errors of the full and pruned networks, which are given in Table 7. In general, the OBD algorithm makes it possible to remove a big portion of weights and may give precise models. In particular, the best models N_{20}^3 and N_{30}^3 , when compared to their full versions, have approximately 60% less weights. It is also interesting to note that in case of the best models the OBD algorithm results in reduction in the validation error by some 30%. The approximation of the validation data set quality using models N_{20}^3 and N_{30}^3 is still almost as good as using unpruned networks. Their comparison with validation data set is depicted in Figs. 15 and 16 for structure N_{20}^3 and N_{30}^3 , respectively.

It is worth noting, that there are models, where it is highly difficult to choose at which iteration the OBD algorithm should be stopped. The computational complexity and the memory that has to be used to store these models can be determined easily based on the model configuration. The complexity is closely related to the number of parameters, whereas the accuracy is difficult to predict. For example, the error values of the model N_{30}^1 slowly grow with each iteration of the OBD algorithm. There are almost only inaccurate and simple models or accurate and complex ones. Almost nothing in between. A similar problem can be faced for the model N_{20}^2 —even though with each removed weight model complexity drops, the accuracy drops as well. On the other hand, there are models for which at the consecutive iterations of the OBD algorithm the model

Table 3 Training (E_t) and validation (E_v) errors for the networks with $K = 20$ hidden nodes after removing the given number of weights

Removed weights	Initial network N_{20}^1		Initial network N_{20}^2		Initial network N_{20}^3	
	E_t	E_v	E_t	E_v	E_t	E_v
10	0.1538	0.7704	0.1710	0.7555	0.1886	0.9096
20	0.1488	0.6015	0.1662	0.6708	0.1901	1.4407
30	0.1554	1.4820	0.1771	0.4896	0.2280	0.6061
40	0.1870	0.8471	0.3548	1.3461	0.1887	0.3792
50	0.1805	0.7399	0.2785	1.7369	0.2390	0.5405
60	0.1978	0.8347	5002.5945	13622.8370	0.3975	1.2945
70	0.2628	2.5504	0.8618	4.1967	0.4308	1.1162
80	0.2720	1.7519	—	—	—	—
90	882.8008	687.5612	—	—	—	—

Table 4 Training (E_t) and validation (E_v) errors for the networks with $K = 30$ hidden nodes after removing the given number of weights

Removed weights	Initial network N_{30}^1		Initial network N_{30}^2		Initial network N_{30}^3	
	E_t	E_v	E_t	E_v	E_t	E_v
10	0.1610	0.5179	0.2249	2.3317	0.1614	0.6359
20	0.1658	0.3658	0.2174	2.2419	0.1567	0.5921
30	0.1583	0.3528	0.2093	1.9629	0.1468	0.4569
40	0.1490	0.2654	0.1996	0.9401	0.1376	0.5026
50	0.1421	0.3178	0.1932	1.0448	0.1355	0.4300
60	0.1729	0.6745	0.1744	1.1082	0.1292	0.3667
70	0.1688	0.5825	1.2428	10.6029	0.1250	0.2689
80	0.1783	0.6548	—	—	0.2060	0.3978
90	0.1755	0.6589	—	—	0.2206	0.4772
100	0.2275	1.2958	—	—	0.3265	0.8097
110	0.2059	2.2145	—	—	7765.5044	7093.5566
120	0.2429	2.8449	—	—	—	—
130	0.9478	8.9890	—	—	—	—

Table 5 Training (E_t) and validation (E_v) errors for the fully pruned networks

Initial network	No. of weights	E_t	E_v
N_{20}^1	26	1086.5534	674.4024
N_{20}^2	44	17458.9152	15841.2064
N_{20}^3	47	11993.3168	161.5995
N_{30}^1	44	2071.2067	1266.7459
N_{30}^2	108	2562.9705	760.8429
N_{30}^3	70	12110.3688	7067.0067

Table 6 Training (E_t) and validation (E_v) errors for the best pruned networks

Initial network	No. of weights	E_t	E_v
N_{20}^1	40	0.2748	1.6894
N_{20}^2	46	0.8542	4.7684
N_{20}^3	52	0.4141	0.9105
N_{30}^1	50	1.2299	5.7530
N_{30}^2	114	0.1695	1.0717
N_{30}^3	80	0.3260	0.8157

simplicity and accuracy grows at the same time. It is because removal of one weight acts like a small disturbance, that helps to leave a local minimum and get closer to a global one. Nevertheless, each of the mod-

Fig. 13 Error values for each sample k , calculated using validation data set and the output of chosen model created by pruning of initial full network: **a** N_{20}^1 , **b** N_{20}^2 , **c** N_{20}^3

els placed in Table 6 is an accurate approximator of the process. It is easily noticed that absolute values of the

Fig. 14 Error values for each sample k , calculated using validation data set and the output of chosen model created by pruning of initial full network: **a** N_{30}^1 , **b** N_{30}^2 , **c** N_{30}^3

Table 7 Percentage of removed weights and percentage ratio of the best pruned networks' training (E_t^p) and validation error (E_v^p) comparing to the initial full networks (E_t^f , E_v^f)

Initial network	Removed weights (%)	$(E_t^p/E_t^f) \times 100\%$ (%)	$(E_v^p/E_v^f) \times 100\%$ (%)
N_{20}^1	66.94	153.95	156.96
N_{20}^2	61.98	447.93	532.19
N_{20}^3	57.02	197.76	73.67
N_{30}^1	72.38	735.59	965.59
N_{30}^2	37.02	67.66	32.73
N_{30}^3	55.80	177.27	68.20

error of each sample for each model (Figs. 9 and 13, 10 and 14) are still very low—the same highest error values occurs before and after pruning.

Figure 5b–d depicts the architectures of the pruned networks with 20 hidden nodes, i.e. the networks N_{20}^1 , N_{20}^2 and N_{20}^3 whereas Fig. 6b–d depicts the pruned networks with 30 hidden nodes, i.e. the networks N_{30}^1 , N_{30}^2 and N_{30}^3 . It is interesting to note that the OBD algo-

Fig. 15 Validation data set (solid line) versus the output of best pruned network N_{20}^3 with 20 hidden nodes (dashed line)

Fig. 16 Validation data set (solid line) versus the output of best pruned network N_{30}^3 with 20 hidden nodes (dashed line)

rithm works in an intelligent way, if all weight in the first layer connected to a some hidden node are deleted, it also removes the corresponding weight in the second layer.

In this work, the OBD algorithm was performed as many times as there were saliency values equal or greater than 0. Knowing that saliency values of removed weights are set to 0, therefore the vector of saliency values at the end of model pruning always consists of negative values as shown in Figs. 17 and 18. Most of time the vector of saliency values consists of nonnegative ones only (Fig. 18), but as in the Fig. 17, after 60 iterations of the OBD algorithm, there are only non-positive values in the saliency vector. The algorithm does not stop there because there are weights for which saliency equals 0, but they are not yet removed—one of them will be removed in the next iteration. At

Fig. 17 Absolute saliency values sorted in descending order (positive values on the left, negative on the right side) during pruning the model N_{20}^2

the iteration no. 95, there are only negative values, and zeros corresponding to removed weights. The saliency of 0 appears when the weight's value or the second-order derivative of the error function with respect to this weight equals 0 as in Eqs. (15) and (16). The second condition takes place when, for example, a hidden neuron has no input signals (i.e. there are no connections between this node and any node in the first layer), then saliency of weight connecting this and summing nodes is 0. It is because it has no influence on the output signal, and so the second-order derivative of the error function equals 0. An analogous case takes place if there are no connections between the first layer's node,

and any of the second layer's node. Worth mentioning is that if saliency equals 0, then that corresponding weight will be removed as fast as possible, what is consistent with intuition—weight linked with node that is of no use, should be removed.

The use of the OBD algorithm requires that the diagonal of Hessian matrix (6) is positive-definite [so that minimum of error function (5) is achieved]. That implies that the saliency values are nonnegative and the removal of weight can be carried out. Computational constraints cause that reaching exact minimum is not always possible, and that cause the saliency values not to be the way we are expecting them to be.

Fig. 18 Absolute saliency values sorted in descending order (positive values on the left, negative on the right side) during pruning the model N_{20}^3

Nonetheless even with this kind of inaccuracy, implementation of OBD algorithm allows to obtain reasonable results.

5 Conclusions

This work describes derivation and implementation details of the OBD algorithm for pruning the recurrent dynamic neural models with one hidden layer. The neutralisation reactor benchmark process is considered to demonstrate effectiveness of the algorithm. The problem resulting from computational inaccuracy is shown,

as well as its possible consequences. The models of two different architectures have been trained and pruned using the discussed implementation of the OBD algorithm. Considering only the best results, for the considered neutralisation process, reduction in the number of weights is approximately 60% and the validation error is some 30% smaller when compared to the full models.

Choosing the model that has a moderate number of parameters and is precise is not a simple task. It requires a compromise between error values and total number of weights of the model. Although this procedure is

time-consuming, it is worth repeating several times to achieve the best model configuration.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Ławryńczuk, M.: Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach. Studies in Systems Decision and Control, vol. 3. Springer, Heidelberg (2014)
- Tatjewski, P.: Advanced Control of Industrial Processes: Structures and Algorithms. Springer, London (2007)
- Korbicz, J., Koscielnny, J.M., Kowalczyk, Z., Cholewa, W.: Fault Diagnosis: Models, Artificial Intelligence, Applications. Springer, London (2004)
- Witczak, M.: Modelling and estimation strategies for fault diagnosis of non-linear systems: from analytical to soft computing approaches. In: Lecture Notes in Control and Information Sciences, vol. 354. Springer, Berlin (2007)
- Witczak, M.: Fault diagnosis and fault-tolerant control strategies for non-linear systems: analytical and soft computing approaches. In: Lecture Notes in Electrical Engineering, vol. 266. Springer, Berlin (2014)
- Simon, D.: Optimal State Estimation: Kalman, H_∞ and Non-linear Approaches. Wiley, Hoboken (2006)
- Luyben, W.L.: Process Modelling, Simulation and Control for Chemical Engineers. McGraw Hill, New York (1990)
- Marlin, T.E.: Process Control. McGraw-Hill, New York (1995)
- Palit, A.K., Popovic, D.: Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications. Springer, Berlin (2005)
- Yan, Z., Wang, J.: Nonlinear model predictive control based on collective neurodynamic optimization. IEEE Trans. Neural Netw. Learn. Syst. **26**, 840–850 (2015)
- Yan, Z., Wang, J.: Robust model predictive control of nonlinear systems with unmodeled dynamics and bounded uncertainties based on neural networks. IEEE Trans. Neural Netw. Learn. Syst. **25**, 457–469 (2014)
- Fortuna, L., Graziani, S., Rizzo, A., Xibilia, M.G.: Soft Sensors for Monitoring and Control of Industrial Processes. Springer, Berlin (2007)
- Ogiela, M., Tadeusiewicz, R.: Modern Computational Intelligence Methods for the Interpretation of Medical Images. Studies in Computational Intelligence, vol. 84. Springer, Heidelberg (2006)
- Nelles, O.: Nonlinear System Identification. From Classical Approaches to Neural Networks and Fuzzy Models. Springer, Berlin (2001)
- Pearson, R.K.: Selecting nonlinear model structures for computer control. J. Process Control **13**, 1–26 (2003)
- Haykin, S.: Neural Networks and Learning Machines. Prentice Hall, Upper Saddle River (2009)
- Ripley, B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge (1996)
- Mandic, D.P., Chambers, J.: Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. Wiley, New York (2001)
- Hosen, M.A., Hussain, M.A., Mjalli, F.S.: Control of polystyrene batch reactors using neural network based model predictive control (NNMPC): an experimental investigation. Control Eng. Pract. **19**, 454–467 (2011)
- Ławryńczuk, M.: Practical nonlinear predictive control algorithms for neural wiener models. J. Process Control **23**, 696–714 (2013)
- Vieira, W.G., Santos, V.M.L., Carvalho, F.R., Pereira, J.A.F.R., Fileti, A.M.F.: Identification and predictive control of a FCC unit using a MIMO neural model. Chem. Eng. Process. **44**, 855–868 (2005)
- De Carvalho, R.M., Mello, C., Kubota, L.T.: Simultaneous determination of phenol isomers in binary mixtures by differential pulse voltammetry using carbon fibre electrode and neural network with pruning as a multivariate calibration tool. Anal. Chim. Acta **420**(1), 109–121 (2000)
- Ghani, N., Lamontagne, R.: Neural networks applied to the classification of spectral features for automatic modulation recognition. In: Military Communications Conference, 1993. MILCOM '93. Conference Record. Communications on the Move, IEEE, vol. 1, pp. 111–115 (1993)
- Giles, C.L., Omlin, C.W.: Pruning recurrent neural networks for improved generalization performance. IEEE Trans. Neural Netw. **5**(5), 848–851 (1994)
- Hassibi, B., Stork, D.G., Wolff, G.J.: Optimal brain surgeon and general network pruning. In: Neural Networks, 1993. IEEE International Conference on, vol. 1, pp. 293–299 (1993)
- Hintz-Madsen, M., Kai Hansen, L., Larsen, J., With Pedersen, M., Larsen, M.: Neural classifier construction using regularization, pruning and test error estimation. Neural Netw. **11**(9), 1659–1670 (1998)
- Le Cun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Touretzky, D. (ed.) Advances in NIPS2, pp. 598–605. Morgan Kaufmann, San Mateo (1990)
- Goh, Y.S., Tan, E.C.: Pruning neural networks during training by backpropagation. In: Proceedings of TENCON'94—1994 IEEE Region 10's 9th Annual International Conference on: 'Frontiers of Computer Technology', vol. 2, pp. 805–808 (1994)
- Mauch, L., Yang, B.: A novel layerwise pruning method for model reduction of fully connected deep neural networks. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2382–2386 (2017)
- Endisch, C., Stolze, P., Endisch, P., Hackl, C., Kennel, R.: Levenberg-marquardt-based OBS algorithm using adaptive pruning interval for system identification with dynamic neural networks. In: Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, pp. 3402–3408 (2009)
- Fog, T.L., Larsen, J., Hansen, L.K.: Training and evaluation of neural networks for multi-variate time series processing. In: Neural Networks, 1995. Proceedings, IEEE International Conference on, vol. 2, pp. 1194–1199 (1995)

32. Huynh, T.Q., Setiono, R.: Effective neural network pruning using cross-validation. In: *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 972–977 (2005)
33. Kaminski, M., Orłowska-Kowalska, T.: Comparison of bayesian regularization and optimal brain damage methods in optimization of neural estimators for two-mass drive system. In: *Industrial Electronics (ISIE)*, 2010 IEEE International Symposium on, pp. 102–107 (2010)
34. Setiono, R., Gaweda, A.: Neural network pruning for function approximation. In: *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 6, pp. 443–448 (2000)
35. Silvestre, M.R., Ling, L.L.: Pruning methods to MLP neural networks considering proportional apparent error rate for classification problems with unbalanced data. *Measurement* **56**, 88–94 (2014)
36. Narendra, K.S., Parthasarathy, K.: Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Netw.* **1**, 4–27 (1990)
37. Shook, D.S., Mohtadi, C., Shah, S.L.: Identification for long-range predictive control. *IEE Proc. D Control Theory Appl.* **138**(1), 75–84 (1991)
38. Shook, D.S., Mohtadi, C., Shah, S.L.: A control-relevant identification strategy for GPC. *IEEE Trans. Autom. Control* **37**(7), 975–980 (1992)
39. Ławryńczuk, M., Tatjewski, P.: Nonlinear predictive control based on neural multi-models. *Int. J. Appl. Math. Comput. Sci.* **20**(1), 7–21 (2010)
40. Ławryńczuk, M.: Training of neural models for predictive control. *Neurocomputing* **73**(7), 1332–1343 (2010)
41. Böling, J.M., Seborg, D.E., Hespanha, J.P.: Multi-model adaptive control of a simulated pH neutralization process. *Control Eng. Pract.* **15**(6), 663–672 (2007)
42. Grancharova, A., Kocijan, J., Johansen, T.A.: Explicit output-feedback nonlinear predictive control based on black-box models. *Eng. Appl. Artif. Intell.* **24**(2), 388–397 (2011)
43. Henson, M.A., Seborg, D.E.: Adaptive nonlinear control of a pH neutralization process. *IEEE Trans. Control Syst. Technol.* **2**(3), 169–182 (1994)
44. Karasakal, O., Guzelkaya, M., Eksin, I., Yesil, E., Kumbasar, T.: Online tuning of fuzzy PID controllers via rule weighing based on normalized acceleration. *Eng. Appl. Artif. Intell.* **26**(1), 184–197 (2013)
45. Kumbasar, T., Eksin, I., Guzelkaya, M., Yesil, E.: Type-2 fuzzy model based controller design for neutralization processes. *ISA Trans.* **51**(2), 277–287 (2012)
46. Oblak, S., Åkrjanc, I.: Continuous-time Wiener-model predictive control of a pH process based on a PWL approximation. *Chem. Eng. Sci.* **65**(5), 1720–1728 (2010)
47. Bonans, J.F., Gilbert, J.C., Lemarechal, C., Sagastizabal, C.A.: *Numerical Optimization: Theoretical and Practical Aspects*. Springer, Berlin (2006)
48. Nocedal, J., Wright, S.J.: *Numerical Optimization*. Springer, Berlin (2006)
49. Ruszczyński, A.: *Nonlinear Optimization*. Princeton University Press, Princeton (2006)
50. Gómez, J.C., Jutan, A., Baeyens, E.: Wiener model identification and predictive control of a pH neutralisation process. *IEE Proc. Part D Control Theory Appl.* **151**, 329–338 (2004)
51. Ławryńczuk, M.: Modelling and predictive control of a neutralisation reactor using sparse support vector machine wiener models. *Neurocomputing* **205**(Supplement C), 311–328 (2016)
52. Yang, Y., Wu, Q.: A neural network PID control for pH neutralization process. In: *2016 35th Chinese Control Conference (CCC)*, pp. 3480–3483 (2016)