A new multilayer perceptron pruning algorithm for classification and regression applications

Philippe Thomas, Marie-Christine Suhner

Neural Process Lett
DOI 10.1007/s11063-014-9366-5

A new multilayer perceptron pruning algorithm for classification and regression applications

Philippe THOMAS[1,2*], Marie-Christine SUHNER[1,2]

[1]*Université de Lorraine, CRAN, UMR 7039, Campus Sciences, BP 70239, 54506 Vandœuvre-lès-Nancy cedex, France*

[2]*CNRS, CRAN, UMR7039, France*

[*]*Corresponding author*

Phone: +33 (0)3 83 68 44 30

Fax: +33 (0)3 83 68 44 59

Email: philippe.thomas@univ-lorraine.fr; marie-christine.suhner@univ-lorraine.fr

Abstract. Optimizing the structure of neural networks remains a hard task. If too small, the architecture does not allow for proper learning from the data, whereas if the structure is too large, learning leads to the well-known overfitting problem. This paper considers this issue, and proposes a new pruning approach to determine the optimal structure. Our algorithm is based on variance sensitivity analysis, and prunes the different types of unit (hidden neurons, inputs, and weights) sequentially. The stop criterion is based on a performance evaluation of the network results from both the learning and validation datasets. Four variants of this algorithm are proposed. These variants use two different estimators of the variance. They are tested and compared with four classical algorithms on three classification and three regression problems. The results show that the proposed algorithms outperform the classical approaches in terms of both computational time and accuracy.

Keywords: neural network, multilayer perceptron, pruning, classification, regression, data mining

Neural Process Lett
DOI 10.1007/s11063-014-9366-5

1 Introduction

From the first works of Rumelhart and McClelland [1], artificial neural network models have been successfully applied to solve many different problems, including the identification of dynamical systems, pattern classification, adaptive control, and function approximation.

Among these artificial neural network models, the multilayer perceptron (MLP) is, by far, the most popular architecture because of its structural flexibility, good representational capabilities, and the availability of a large number of training algorithms [2]. This model is used for both classification and regression tasks. The goal of classification is to map the data into predefined groups or classes. In the regression phase, the aim is to map data items to a real-valued prediction variable [3].

For both classification and regression, the same question must be addressed: how many nodes should be used in each layer? Despite the wealth of literature on the MLP, defining the optimal architecture remains a hard task. If too small, the architecture cannot learn from the data properly, whereas if the structure is too large, learning leads to the well-known overfitting problem. To avoid overfitting, we can apply early stopping [4], a robust minimization criterion [5-6], regularization approaches [7-9], or determine the optimal structure of the network. The latter method allows us to obtain a simpler and smaller model. Hence, a crucial step in designing the MLP is network model selection [10]. Many algorithms have been proposed to determine the optimal architecture of the network. However, because this determination is an NP-hard problem, most existing algorithms are approximate and produce near-optimal solutions [11]. These algorithms can be classified into four groups: pruning algorithms, constructive algorithms, hybrid algorithms, and evolutionary algorithms. Pruning algorithms learn using an over-sized structure, and then, in a second step, eliminate spurious parameters and/or neurons. One of the main advantages of these algorithms is that they are able to perform feature selection simultaneously [12-16]. Constructive algorithms start from a small initial architecture, and add new hidden units or layers until the results are acceptable [17-20]. Hybrid algorithms combine pruning and constructive approaches [21-23], whereas

evolutionary algorithms conduct a search across the parameter space and determine the optimal structure using some performance index [24-25].

This paper focuses on pruning algorithms, and proposes a new method based on the variance sensitivity analysis proposed by Engelbrecht [26]. We prune different types of unit (hidden neurons, inputs, and weights) sequentially, and use a stop criterion based on a performance evaluation of the network results from both the learning and validation datasets. The main goals of this algorithm are to improve structure determination and minimize computational time for both classification and regression. Four variants of this algorithm are proposed.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of structure determination and pruning algorithms, and Section 3 describes the network architecture and learning algorithm. Section 4 introduces the proposed pruning algorithm and its four variants. Sections 5 and 6 present the results obtained from benchmark classification and regression tasks, respectively. The results given by the proposed algorithms are compared with those obtained by conventional pruning algorithms. Finally, Section 7 concludes the paper.

2 Brief overview of structure determination

2.1 Problem origination

As explained previously, determining the optimal architecture of a neural network is an NP-hard problem. The original work on this problem is Kolmogorov's theorem, which states that any continuous function on an n-dimensional cube can be represented by the sum and superposition of continuous functions of one variable [27]. Cybenko [28] and Funahashi [29] have proved that sigmoidal functions are suitable for this work, and so neural networks with only one hidden layer that use a sigmoidal function can be seen as universal approximators. However, these results are not sufficient to determine the size of this hidden layer. Hecht-Nielsen [30] has proved that any continuous function can be represented by a neural network that has only one hidden layer with exactly $2n_i+1$ neurons, where n_i is the number of inputs. However, the activation function used in his work is more complex than the sigmoidal one. Huang [31] has extended this work, and proved that, with two hidden layers, $2\sqrt{(n_o+2)N}$ hidden units allocated between the two hidden layers are enough to learn N samples with negligibly small error

(n_o denotes the number of outputs). The problem here is to fit a function with a finite number of (generally noisy) samples. In this case, the network may learn the noise on the function, and not the function itself, leading to an overfitting problem [32].

Thus, no exact paradigm exists to estimate the optimal neural network, with different approaches and algorithms being used to determine its structure.

2.2 Optimal structure determination approaches: a brief overview

The oldest and most primitive approach to determining the optimal structure is simple trial-and-error, which can lead to severely suboptimal structures.

Another approach is to perform an exhaustive search. However, this is not a realistic option for many real applications. Moreover, an exhaustive search will be complicated by the noisy fitness evaluation problem [33].

2.2.1. Evolutionary approach

Evolutionary algorithms can be seen as an extension of the exhaustive search using mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Evolutionary approaches have been used in a restrictive way to prune oversized structures [34-35]. However, this approach may be used with a hybrid goal (construction and pruning) [36-37]. Evolutionary algorithms may be used to simultaneously optimize the architecture and weights [25, 38-39] or to concentrate on the structure [35-37, 40].

The use of evolutionary algorithms for structure determination introduces two sources of noise, one from the initialization of weights, and the other from the chosen optimizer. In contrast, the simultaneous evolution of architecture and weights is muddled by the well-known permutation problem. Additionally, the computational cost of implementing evolutionary algorithms, even for structure determination alone, may become prohibitive [41].

2.2.2. Constructive approach

The goal of constructive approaches is to determine the optimal structure of the network by sequentially adding hidden neurons or hidden layers to a small initial structure. Many algorithms have been proposed under this paradigm [42]. Among these algorithms, we can cite dynamic node creation and its variants [43-44],

activity-based structure-level adaptation [45-46], cascade correlation [18, 47], and the constructive one-hidden-layer algorithm and its variants [42, 48-49]. Most of these algorithms lead to more complex structures by mixing various activation functions [20, 49] or adding new hidden layers with connections between neurons that are not in successive layers [18, 41]. Hence, these algorithms lead to neural network structures that are not strictly feedforward, and require specific algorithms to learn parameters while the structure grows [48, 50]. Moreover, these algorithms can lead to suboptimal networks [42], or become trapped in local minima because the error surface of a smaller network is more complicated and includes more local minima than the error surface of a larger network [51-52].

2.2.3. Pruning approach

In pruning approaches, an oversized structure is first learned, before spurious parameters are eliminated in a second step. Many algorithms using this approach have been proposed, and these can be classified into three broad groups: magnitude-based (pruning during learning), brute-force, and sensitivity-based (post-learning pruning) [15-16].

In magnitude-based approaches (pruning during learning), a term is added to the objective function for choosing the most efficient solution. For example, a term proportional to the sum of all weight magnitudes favors a solution with small weights; those that are nearly zero are not likely to influence the output much, and can thus be eliminated [15]. This approach prunes the network by driving the weights toward zero during learning, and various different penalty terms and weight decays have been proposed and studied [9, 53-58].

Brute-force pruning methods (post-learning pruning) set the weights to zero one-by-one, and evaluate the change in the error. If it increases too much, then that weight is restored; otherwise, the element is removed. Setiono and Leow [59] used this approach to prune hidden neurons and inputs in two steps.

Sensitivity-based approaches (post-learning pruning) evaluate the sensitivity of the error function to the removal of an element. Some algorithms consider the change in the error function due to small changes in the values of the weights. Measures of the relative effect of the different weights, or saliencies, can be computed, and weights with low saliencies are deleted. Optimal Brain Damage [13], Optimal Brain Surgeon (OBS) [14], and all algorithms derived from these

[60-61] use a second-order Taylor expansion of the error function to estimate how the training error will change as the weights are perturbed. Leung *et al*. [62] applied the same approach for a recursive least-squares algorithm, whereas Tang *et al*. [63] used the error covariance matrix obtained during learning in a similar way to the Hessian matrix in the OBS algorithm.

The output of sensitivity analysis methods is based on a variance analysis of sensitivity information, given by the derivative of the neural network output with respect to the parameters [26]. These are powerful methods, because the network structure inherently contains all of the information needed to compute these derivatives efficiently [64]. Zeng and Yeung [65] inserted an input perturbation and studied its effect on the output sensitivity. Chandrasekaran *et al*. [66] proposed a sensitivity-based method based on linear unit models, and Lauret *et al*. [67] used the extended Fourier amplitude test to quantify the relevance of the hidden neurons. Augasta and Kathirvalavakumar [68] defined the significance of a neuron as the sum-norm of its output. They then compared this significance to a threshold to determine which neurons must be pruned.

Levin *et al*. [69] applied principal component analysis to select elements for removal, while Medeiro and Barreto [70] determined which weights to prune by evaluating the correlation between error signals associated with the neurons of a given layer and the error signal propagated back to the previous layer. Weights that generate lower correlations are not relevant, and can therefore be pruned. Beigy and Meybodi [11] proposed an automaton that can determine whether a hidden neuron should be preserved or pruned. This automaton exploits the variance of the activation of a neuron. Liang [71] developed an orthogonal projection to determine the importance of hidden neurons.

Although there are many different pruning methods, the main ideas underlying most of them are similar. They all try to establish a reasonable relevance measure for a specified parameter (input, weight, or neuron), so that the pruning action will hopefully have the least effect on the performance of the network [65].

One of the disadvantages of pruning algorithms is their heavy computational burden, as the majority of the training time is spent on networks that are larger than necessary [2].

2.2.4. Hybrid approach

Hybrid algorithms combine pruning and constructive approaches. Nabhan and Zomaya [22] exploited an intelligent generate-and-test procedure that evaluates the learning performance of the structure, and then modifies it by exploring different alternatives and selecting the most promising. The algorithm modifies the structure of the neural network by adding or deleting neurons and/or layers. Rivals and Personnaz [72] associated a constructive approach based on conditioning the Jacobian matrix of the candidate neural models with a pruning phase based on statistical tests. Huang *et al*. [73] have developed an on-line training algorithm for a radial basis function neural network, and Narasimha *et al*. [23] reported a method that starts with an empty network (no hidden unit) and successively adds hidden units, retraining only these new units. In a second step, an orthonormalization pruning algorithm is used to remove spurious parameters. Hsu [74] has exploited an on-line hybrid algorithm to design a neural network controller for a linear piezo-electric ceramic motor. Han and Qiao [2] optimized the network structure by combining error reparation and sensitivity analysis techniques.

The main drawbacks of these algorithms are similar to those of the constructive approach; i.e., they may require particular learning algorithms and can lead to suboptimal networks [52].

2.3. Selection criteria

The main risk is that current algorithms may become stuck at local minima. Thus, different sets of initial weights are generally used to avoid the local minima trapping problem [11]. In addition, selection criteria are generally used to determine a suitable structure.

For regression problems, many selection criteria have been proposed, such as Akaike's information criterion (AIC), Bayesian information criterion (BIC), root mean square error (RMSE), and mean absolute percentage error (MAPE). Qi and Zhang [75] compared these criteria and their extensions, and concluded that there is no best method. Egrioglu *et al*. [76] proposed an association of these different criteria to select the optimal structure. AIC and BIC penalize large models, whereas RMSE and MAPE are measures of the deviation of the predicted values from actual values [76]. For regression problems, the goal is to reduce the

distance between the model and the data. Hence, the RMSE criterion is suitable, and is used in this work:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2} \quad (1)$$

where N denotes the number of data, y_n is the n^{th} actual data point, and $\hat{y}_n$ is the predicted value.

In classification problems, the goal is to reduce the number of misclassified data. Thus, the classical criterion for classification problems is the misclassification rate (error rate or "zero-one" score function) [77]:

$$S_{01} = \frac{1}{N}\sum_{n=1}^{N} I(y_n, \hat{y}_n) \quad (2)$$

where $I(a, b) = 1$ when $a \neq b$ and 0 otherwise.

A related difficulty arises if there are different costs associated with different misclassifications. In this case, a general loss matrix should be constructed [10]. This problem is not considered here.

3 Network architecture and learning algorithm

3.1. Multilayer perceptron

The works of Cybenko [28] and Funahashi [29] have shown that a multilayer neural network consisting of only one hidden layer (using a sigmoidal activation function) and an output layer can approximate all nonlinear functions with the desired accuracy. This result explains the great interest in this type of neural network, the aforementioned MLP.

Its structure is given by:

$$\hat{y} = g_o\left(\sum_{h=1}^{n_o} w_h^2 . g_h\left(\sum_{i=1}^{n_i} w_{hi}^1 . x_i + b_h^1\right) + b\right) \quad (3)$$

where x_i are the n_i inputs of the neural network, w_{hi}^1 are the weights connecting the input layer to the hidden layer, b_h^1 are the biases of the hidden neurons, $g_h(.)$ is the activation function of the hidden neurons (here, the hyperbolic tangent for all hidden neurons), w_h^2 are the weights connecting the hidden neurons to the

output neuron, b is the bias of the output neuron, $g_o(.)$ is the activation function of the output neuron, and $\hat{y}$ is the network output.

The activation function used for the output layer differs according to the problem under consideration. For regression problems, this function is typically chosen to be linear, whereas for classification problems, the network's output should represent the probability that a pattern belongs to the considered class. In this case, the activation function $g_o(.)$ is generally chosen to be sigmoidal.

3.2. Learning algorithm

The MLP's parameters (weights and biases) are determined using supervised learning. This performs a local search for the optimum, which implies that the initial set of parameters has a great influence on the final result.

3.2.1. Weights and biases initialization

Many initialization algorithms have been proposed [78]. We use a modification of the Nguyen and Widrow (NW) algorithm [79], which allows a random initialization of weights and biases to be associated with an optimal placement in the input and output spaces. The NW algorithm has been modified by Demuth and Beale [80] for non-normalized inputs.

For regression problems, the output of the network may be a real value if no normalization occurs. To take this into account, a second modification has been introduced concerning the biases and weights connecting the hidden layer to the output layer to deal with non-normalized outputs [78].

3.2.2. Robust learning algorithm

We use a Levenberg–Marquardt algorithm associated with a robust criterion [6]. This algorithm performs a local search for the optimum, and the robust criterion accounts for the presence of outliers in the data and has a regularization effect during learning. The robust criterion is based on Huber's model of measurement noise contaminated by outliers [81]. This model considers the distribution of the noise e as a mixture of two density functions. The first corresponds to the basic distribution of a measurement noise (e.g., normal, variance σ_1^2), whereas the second, corresponding to outliers, is an arbitrary symmetric long-tailed distribution (e.g., normal, but with variance σ_2^2 such that $\sigma_1^2 < \sigma_2^2$).

4 Proposed pruning algorithm

Even if the learning algorithm has a regularization effect, it is useful to determine the optimal structure of the network to simplify the model.

4.1. Variance nullity measure

The proposed algorithm uses the variance nullity measure (VNM) [26, 82-83], in which the variance of the sensitivity of an input, a hidden neuron's output, or a parameter (weight or bias) is measured for different patterns. If this variance is not significantly different from zero, and if the average sensitivity is small, the input or the hidden neuron under consideration has no effect on the network output. Therefore, the VNM can be used in hypothesis testing to determine whether an input, hidden neuron, or parameter has a statistical impact on the network using the χ^2 distribution. If not, it must be pruned. This is repeated for each type of element (input, hidden neuron, weight connecting input to hidden layer, bias of hidden layer), and requires the sensitivity of the network output to be determined for each type of element.

4.1.1. Sensitivity to a hidden neuron

To determine whether a hidden neuron *h* must be pruned, the VNM of the weight w_h^2 ($h = 1 \ldots n_h$) that connects this hidden neuron to the output neuron must be calculated. For this, we must determine the sensitivity of the network output $\hat{y}$ to the parameter w_h^2. This sensitivity corresponds to the contribution of this parameter to the output error. This contribution is determined by the partial derivative of the network output $\hat{y}$ with respect to the parameter w_h^2 being considered:

$$S_{w_h^2}(n) = \frac{\partial \hat{y}(n)}{\partial z(n)} . \frac{\partial z(n)}{\partial w_h^2} = g_o^{'}(z(n)).x_h^1(n) \qquad (4)$$

where *n* is the number of data patterns from the learning database and $g_o^{'}(z(n))$ is the derivative of the activation function of the output neuron $g_o()$. When this activation function is linear (i.e., for regression problems), its derivative is:

$$g_o^{'}(z(n)) = 1 \qquad (5)$$

Neural Process Lett
DOI 10.1007/s11063-014-9366-5

For classification problems, the activation function is chosen to be sigmoidal, and its derivative becomes:

$$g_o^{'}(z(n)) = \hat{y}(n).(1-\hat{y}(n)) \quad (6)$$

4.1.2. Sensitivity to an input

Similarly, the sensitivity of the network output $\hat{y}$ to the input x_i (i=1…n_i) is obtained by taking the partial derivative of the output with respect to the input x_i under consideration:

$$\begin{aligned} S_{x_i}(n) &= \frac{\partial \hat{y}(n)}{\partial x_i(n)} = \sum_{h=1}^{n_h} \frac{\partial \hat{y}(n)}{\partial z(n)} \cdot \frac{\partial z(n)}{\partial x_h^1(n)} \cdot \frac{\partial x_h^1(n)}{\partial z_h^1(n)} \cdot \frac{\partial z_h^1(n)}{\partial x_i(n)} \\ &= \sum_{h=1}^{n_h} g_o^{'}(z(n)).w_h^2.\left(1-\left(x_h^1(n)\right)^2\right).w_{hi}^1 \end{aligned} \quad (7)$$

where $g_o^{'}(z(n))$ is given by (5) for regression problems and (6) for classification problems.

4.1.3. Sensitivity to a weight connecting the input to the hidden layer

The sensitivity of the network output $\hat{y}$ to the weights w_{hi}^1 (i = 1..., n_i; h = 1..., n_h) connecting the input to the hidden layers is obtained by taking the partial derivative of the output with respect to the weight w_{hi}^1 under consideration:

$$\begin{aligned} S_{w_{hi}^1}(n) &= \frac{\partial \hat{y}(n)}{\partial w_{hi}^1(n)} = \sum_{h=1}^{n_h} \frac{\partial \hat{y}(n)}{\partial z(n)} \cdot \frac{\partial z(n)}{\partial x_h^1(n)} \cdot \frac{\partial x_h^1(n)}{\partial z_h^1(n)} \cdot \frac{\partial z_h^1(n)}{\partial w_{hi}^1(n)} \\ &= \sum_{h=1}^{n_h} g_o^{'}(z(n)).w_h^2.\left(1-\left(x_h^1(n)\right)^2\right).x_i(n) \end{aligned} \quad (8)$$

where $g_o^{'}(z(n))$ is given by (5) for regression problems and (6) for classification problems.

4.1.4. Sensitivity to the bias of the hidden layer

The sensitivity of the network output $\hat{y}$ to the bias b_h^1 (h = 1..., n_h) of the hidden neurons is obtained by taking the partial derivative of the output with respect to the bias b_h^1 under consideration:

$$S_{b_h^1}(n) = \frac{\partial \hat{y}(n)}{\partial b_h^1(n)} = \sum_{h=1}^{n_h} \frac{\partial \hat{y}(n)}{\partial z(n)} . \frac{\partial z(n)}{\partial x_h^1(n)} . \frac{\partial x_h^1(n)}{\partial z_h^1(n)} . \frac{\partial z_h^1(n)}{\partial b_h^1(n)} = \sum_{h=1}^{n_h} g_o^{'}(z(n)).w_h^2.\left(1-\left(x_h^1(n)\right)^2\right) \quad (9)$$

where $g_o^{'}(z(n))$ is given by (5) for regression problems and (6) for classification problems.

4.1.5. Determination of the VNM

The sensitivity of the network output to a hidden neuron, input, weight, or bias of the hidden layer can be explained using the unified notation $S_{\theta_k}(n)$ $\{n = 1\ldots N$ and $k = 1\ldots K=n_i+n_h.(n_i+2)\}$, with θ_k corresponding to x_i when considering input i, to w_h^2 when considering hidden neuron h, to w_{hi}^1 when considering the weight connecting input i to hidden neuron h, or to b_h^1 when considering the bias of hidden neuron h. Thus, $S_{\theta_k}(n)$ is given by equation (4), (7), (8), or (9).

The VNM is the unknown variance $\sigma_{\theta_k}^2$ of the sensitivity of output to parameter θ_k. Engelbrecht [26] used a classical estimator for this variance:

$$\hat{\sigma}_{\theta_k}^2 = \frac{\sum_{n=1}^{N}\left(S_{\theta_k}(n) - \overline{S_{\theta_k}}\right)^2}{N-1} \quad (10)$$

where $\overline{S_{\theta_k}}$ is the mean of the sensitivity of the output to θ_k:

$$\overline{S_{\theta_k}} = \frac{\sum_{n=1}^{N} S_{\theta_k}(n)}{N} \quad (11)$$

Another estimation of this variance is given by [84]:

$$\hat{\sigma}_{\theta_k}^2 = \frac{MAD}{0.7} \quad (12)$$

where MAD is the median of $\left\{\left|S_{\theta_k}(n) - \tilde{S}_{\theta_k}\right|\right\}$, with $\tilde{S}_{\theta_k}$ the median of $S_{\theta_k}(n)$.

This estimator uses the median instead of the mean, and is more robust to the presence of outliers in the data. These two variants are tested and compared in later sections.

A statistical hypothesis test is used to determine whether the VNM of the considered parameter is null. The null hypothesis $\mathcal{H}_0$ (that the variance in parameter sensitivity is approximately zero) and its alternative $\mathcal{H}_1$ are:

$$\begin{cases} \mathcal{H}_0 : & \sigma_{\theta_k}^2 \leq \sigma_0^2 \\ \mathcal{H}_1 : & \sigma_{\theta_k}^2 > \sigma_0^2 \end{cases} \qquad (13)$$

where σ_0^2 is a small positive real number.

Using the fact that, under the null hypothesis, the relation:

$$\Gamma_{\theta_k} = \frac{(N-1).\hat{\sigma}_{\theta_k}^2}{\sigma_0^2} \qquad (14)$$

has a $\chi^2(v)$ distribution, with $\upsilon = N-1$ degrees of freedom in the case of N patterns, the test described by (13) is performed by comparing (14) with the critical value Γ_c obtained from χ^2 distribution tables:

$$\Gamma_C = \chi^2(\upsilon, 1-\alpha) \qquad (15)$$

where α is the significance level of the test. If $\Gamma_{\theta_k} \geq \Gamma_C$, the element under consideration is retained; if not, it is pruned.

The value of σ_0^2 is crucial to the success of this test. If σ_0^2 is too small, no elements will be pruned, whereas if σ_0^2 is too large, too many inputs or hidden neurons will be discarded. Engelbrecht [26] advises starting with a small value of σ_0^2 (0.001) and multiplying this value by 10 if nothing is pruned, up to a maximal value of $\sigma_{\max}^2 = 0.1$.

4.2. Neural network sequential pruning algorithm (NNSP)

The proposed NNSP pruning algorithm uses the VNM described above. The main idea of this algorithm is to evaluate and sequentially prune, if necessary, the different types of elements. Our aim is to improve both the computational time and pruning results. Two variants of this algorithm are proposed, and these will be compared in the two next sections.

4.2.1. NNSP hidden neurons before inputs (NNSP-HI)

This algorithm consists of three steps. In this version, these are the pruning of the hidden neurons, pruning of the inputs, and pruning of the weights and biases of the hidden layer. This algorithm is described in Figure 1.

In each step, two stop criteria are used. The first is when σ_0^2 reaches its maximal value $\sigma_{\max}^2$. The second is based on the mean absolute deviation (MAD), which measures the performance of the network [59]. Two MAD values are calculated from the learning dataset (subscript *T*) and the validation dataset (subscript *V*):

$$\begin{cases} M_T = \dfrac{1}{N_T}.\sum_{n=1}^{N_T}\left|y_T(n)-\hat{y}_T(n)\right| \\ M_V = \dfrac{1}{N_V}.\sum_{n=1}^{N_V}\left|y_V(n)-\hat{y}_V(n)\right| \end{cases} \quad (16)$$

The algorithm is initialized by calculating M_T and M_V (16), and by initializing the memories $M_T^{best} = M_T$, $M_V^{best} = M_V$ and a threshold $Er_{\max} = \left(\max\left\{M_T^{best}; M_V^{best}\right\}\right) * (1+\delta)$, where δ (tuned to 0.025) is used to avoid an early termination to the pruning.

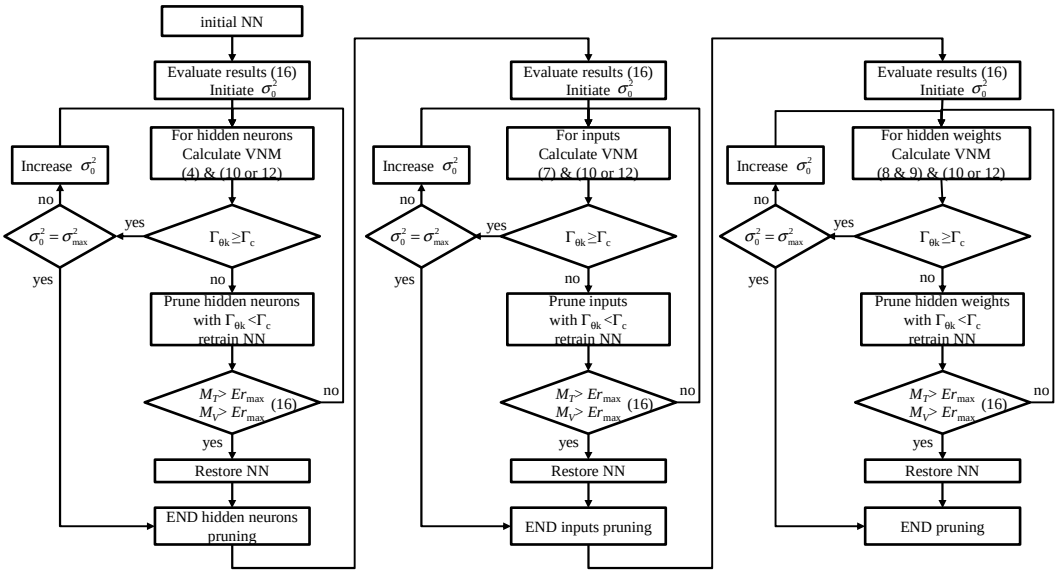

Fig. 1 NNSP-HI algorithm

The pruning phase of the hidden neurons starts by learning the oversized MLP. First, we initialize σ_0^2, the MAD values M_T and M_V, and the threshold $Er_{\max}$. The algorithm then calculates the VNM measures of all the hidden neurons. Two

variants of these VNM measures are determined using equations (4) and (10) (NNSP-HI-mean) or equations (4) and (12) (NNSP-HI-mad). The values of Γ_{θ_k} for all hidden neurons are compared to the threshold Γ_c. If none of the Γ_{θ_k} are lower than this threshold, the parameter σ_0^2 is multiplied by 10 and the Γ_{θ_k} values are recalculated.

If some of the Γ_{θ_k} measures are lower than the threshold Γ_c, the corresponding hidden neurons are pruned and the network is retrained. The two MAD values are calculated from (16), and compared to the threshold $Er_{\max}$. If the new structure improves the results, the memories M_T^{best}, M_V^{best}, and the threshold $Er_{\max}$ are updated with their improved values. If the resulting network becomes too degraded, the preceding structure is restored and the hidden neuron pruning phase terminates. The input pruning phase can then start. The pruning phases of the inputs and hidden weights and biases takes the same approach as the hidden neuron pruning phase, using equation (7) and equations (8) and (9), respectively.

4.2.2. NNSP inputs before hidden neurons (NNSP-IH)

The second version of the algorithm first prunes the inputs, before eliminating spurious hidden neurons and, finally, the weights and biases of the hidden layer. This algorithm is described in Figure 2, and is very similar to the first algorithm. As in the first algorithm, two variants of the VNM measures are determined using equation (10) or equation (12). These two variants of the algorithms are denoted as NNSP-IH-mean and NNSP-IH-mad, respectively.

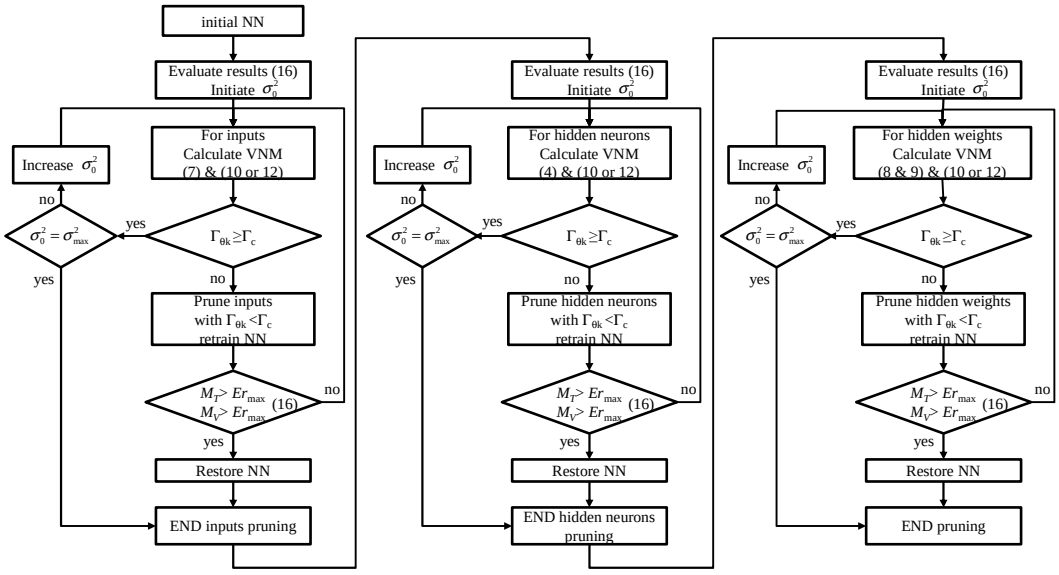

Fig. 2 NNSP-IH algorithm

5 Experimental results on classification examples

In this section, the proposed algorithms (four variants) are implemented and compared with the previous pruning methods of VNP [26], N2PFA [59], OBS [14], and N2PS [68] on three classification problems. For each problem, 20 initial parameter sets are constructed (§3.2.1) and learned with the robust algorithm (§3.2.2). The eight pruning algorithms are applied to the same 20 learned networks.

5.1. Pima Indians Diabetes dataset

This dataset is available from WEKA [85-86]. The problem is to predict whether a patient will test positive or negative for diabetes according to the criteria given by the World Health Organization. This is a two-class problem with class values of 0 (500 data) and 1 (268 data) interpreted as negative and positive results for diabetes. There are eight input data for each pattern. This dataset was randomly divided into two datasets for learning (375 data) and validation (393 data). The initial learning was performed with 20 hidden neurons.

Table 1 lists the best results obtained with the eight pruning algorithms, as well as for the learning without pruning. These results show that even when learning is performed on an oversized structure, the use of a robust criterion allows the generalization capabilities of the network to be preserved.

The best misclassification rate $S_{01_{\min}}$ was obtained on the validation dataset by the NNSP-IH-mean algorithm. A McNemar statistical hypothesis test was used to determine whether the misclassification rate of other algorithms was statistically different to that obtained with NNSP-IH-mean. The null hypothesis (that the tested algorithms is statistically equal to the best one) was tested, where $\mathcal{H}_0$ and its alternative $\mathcal{H}_1$ are:

$$\begin{cases} \mathcal{H}_0 : & S_{01} = S_{01_{\min}} \\ \mathcal{H}_1 : & S_{01} \neq S_{01_{\min}} \end{cases} \qquad (17)$$

The null hypothesis $\mathcal{H}_0$ is rejected with a risk level of 5% if:

$$U = \frac{|N_{10} - N_{01}|}{\sqrt{N_{10} + N_{01}}} > 1.96 \qquad (18)$$

where N_{10} is the number of cases where the best classifier gives correct class and the compared classifier gives wrong class and N_{01} is the number of cases where the best classifier gives wrong class and the compared classifier gives correct class. The results of this test are presented in Table 1. They show that the N2PS and VNP algorithms become trapped at a poor local minimum.

The four proposed algorithms outperformed N2PS and VNP, and gave similar results to the N2PFA and OBS algorithms on the validation dataset. However, the four proposed algorithms pruned more parameters and inputs than N2PFA and OBS without degrading the results, and were less time-consuming (requiring ≈50% of the time of N2PFA). The smallest structure was given by the NNSP-HI-mean algorithm.

Table 1: Best results obtained on the diabetes dataset

	Structure of NN				Misclassification rate S_{01}		U
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	Validation
without pruning	8	20	201	-	0.221	0.247	0.80
NNSP-HI-mean	7	3	28	0.45s	0.189	0.237	0.25
NNSP-HI-mad	7	10	74	0.41s	0.192	0.237	0.28
NNSP-IH-mean	7	11	60	0.851s	0.149	0.232	-
NNSP-IH-mad	7	12	87	0.41s	0.200	0.242	0.53
N2PFA	8	11	111	1.15s	0.112	0.244	0.60
N2PS	5	10	71	0.17s	0.272	0.298	2.80
OBS	8	20	131	48.34s	0.253	0.237	0.50
VNP	7	15	136	0.02s	0.237	0.303	2.77

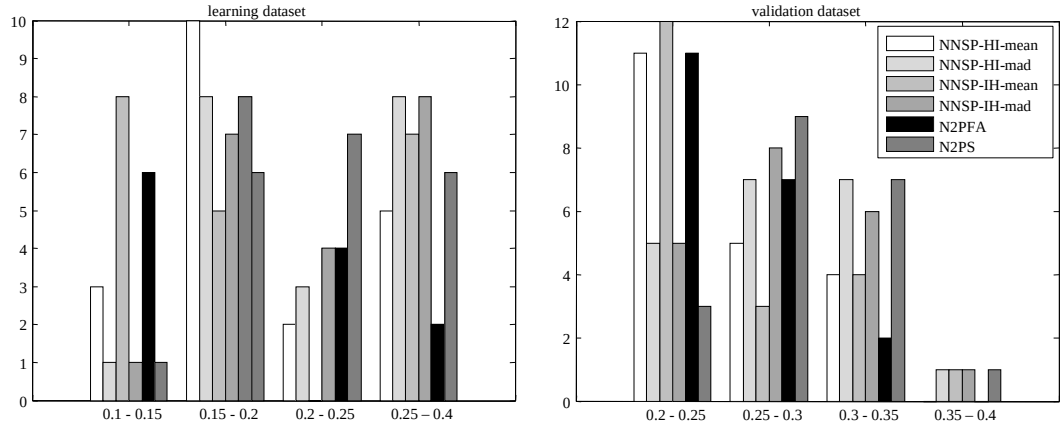

Fig. 3 Distribution of the misclassification rate on learning and validation datasets

Figure 3 presents the distribution of the misclassification rate given by the six best pruning algorithms. These results were obtained over the 20 networks on the learning and validation datasets. This figure shows that NNSP-HI-mean, NNSP-IH-mean, and N2PFA often produced good results. In particular, on the validation

dataset, the misclassification rate given by these three algorithms belongs to the best class (0.2–0.25) in more than 50% of cases.

5.2. Ionosphere dataset

The second classification problem considered the ionosphere dataset, also available from WEKA [86-87]. These radar data were collected by a system in Goose Bay, Labrador, Canada. This is a two-class problem with class values of 0 (225 data) and 1 (126 data) interpreted as bad and good. There are 33 input data for each pattern. The dataset was randomly divided into two datasets, one for learning (186 data) and the other for validation (165 data). The initial learning was performed with 10 hidden neurons.

Table 2 shows the best results obtained with the eight pruning algorithms and for the learning without pruning. As for the diabetes dataset, the use of a robust criterion allowed the generalization capabilities of the network to be preserved. The OBS algorithm produced the best results on the validation dataset, albeit with a considerable computation time. The other algorithms gave equivalent results for the misclassification rate on the validation dataset. This is confirmed by the results of the statistical hypothesis test (17), which are always less than 1.96. However, three of the four proposed algorithms reached these results with a smaller structure than that of OBS (NNSP-HI-mean; NNSP-HI-mad; NNSP-IH-mad).

Table 2: Best results obtained on the ionosphere dataset

	Structure of NN				Misclassification rate S_{01}		U
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	Validation
without pruning	33	10	351	-	0.0000	0.1271	0.99
NNSP-HI-mean	26	2	57	0.25s	0.0647	0.1492	1.65
NNSP-HI-mad	30	10	53	0.63s	0.0000	0.1215	0.80
NNSP-IH-mean	26	10	198	0.75s	0.0000	0.1492	1.55
NNSP-IH-mad	17	10	50	0.70s	0.0765	0.1215	0.76
N2PFA	31	7	232	1.04s	0.0294	0.1326	1.11
N2PS	24	5	131	0.17s	0.0000	0.1436	1.54
OBS	25	8	58	157.2s	0.0529	0.0884	-
VNP	32	10	341	0.01s	0.0706	0.1271	0.83

It can be seen that, in many cases, the pruning algorithms were unable to find a better structure than the initial one. N2PFA and the four proposed algorithms did not prune a single parameter in 30% of the cases, and N2PS retained an unaltered structure in 75% of cases.

5.3. Quality dataset

The third classification problem considered is a real quality classification problem from a robotic lacquering process [88]. In this dataset, one defect (stains on back) is considered. This is a two-class problem with class values of 0 (1997 data) and 1 (273 data) interpreted as having no defects and being defective, respectively. There are 15 input data for each pattern. The dataset was randomly divided into two for learning (1099 data) and validation (1171 data). The initial learning was performed with 25 hidden neurons.

Table 3 shows the best results obtained with the eight pruning algorithms and for the learning without pruning.

Table 3: Best results obtained on the quality dataset

	Structure of NN				Misclassification rate S_{01}		U
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	Validation
without pruning	15	25	426	-	0.1605	0.1448	2.71
NNSP-HI-mean	10	25	93	24min 54s	0.1260	0.1145	0.60
NNSP-HI-mad	10	15	53	57min 44s	0.1338	0.1072	-
NNSP-IH-mean	9	25	81	58min 36s	0.1231	0.1164	0.80
NNSP-IH-mad	15	8	137	57min 3s	0.1307	0.1159	0.71
N2PFA	14	23	369	56 min 3s	0.1449	0.1372	2.11
N2PS	10	12	145	57min 39s	0.1524	0.1390	2.24
OBS	13	25	178	1h 12min 39s	0.1157	0.1247	1.35
VNP	8	25	196	56min 25s	0.1303	0.1626	3.80

The results of the statistical hypothesis test (17) show that N2PFA, N2PS, and VNP gave statistically worse results than those obtained by the best algorithm, NNSP-HI-mad. However, the use of a robust criterion during learning was not sufficient to avoid the overfitting problem.

Moreover, the four variants of the proposed algorithm outperformed the other algorithms on the validation dataset and produced the smallest structures. Note that the VNP algorithm gave acceptable results in only one case. All other algorithms gave acceptable results in over 50% of cases. NNSP-HI-mean and NNSP-IH-mean, which use the variance estimation in (10), were unable to prune any of the hidden neurons. The smallest structure was given by NNSP-HI-mad.

6 Experimental results on regression examples

In this section, the proposed algorithms (four variants) are compared with VNP [26], N2PFA [59], OBS [14], and N2PS [68] on three regression problems. For each problem, 20 initial sets of parameters are constructed (§3.2.1) and learned

with the robust algorithm (§3.2.2). The eight pruning algorithms are applied to the same 20 learned networks.

6.1. Modeling of a static system

The problem considered here is the modeling of a simple nonlinear system [89]. This system is based on a one-hidden-layer perceptron structure with three inputs and one output. This system, supposed to be unknown, is chosen to avoid problems related to the differences between the form of the 'true' model and that of the fitted model. The system is described by:

$$y(t) = 1 + \tanh\left(2.x_1(t) - x_2(t) + 3.x_3(t)\right) + \tanh\left(x_2(t) - x_1(t)\right) + e(t) \quad (19)$$

where *e(t)* is additive Gaussian noise with mean 0 and standard deviation 0.2. Two datasets of 500 points were created for learning and validation. These two datasets include five input variables (x_1, x_2, and x_3 and two supplementary ones). These five inputs are sequences of steps of random length and amplitude. To give each input a different influence, input ranges of [−1; 1], [0; 1.5], [−1; 1.5], [0; 0.5], and [−1; 0] were applied, respectively. The initial structure of the neural network comprised eight hidden neurons.

Table 4 lists the best results obtained with the eight pruning algorithms and for the learning without pruning.

In regression, the mean of the residual must be null. Thus, a two-tailed statistical hypothesis test can be employed to determine this for the different algorithms. The null hypothesis $\mathcal{H}_0$ (that the mean of the residuals is null) and its alternative $\mathcal{H}_1$ are:

$$\begin{cases} \mathcal{H}_0 : & \mu = 0 \\ \mathcal{H}_1 : & \mu \neq 0 \end{cases} \quad (20)$$

where μ is the mean of the residuals population. $\mathcal{H}_0$ is rejected with a risk level of 5% if:

$$\begin{cases} U = \dfrac{\bar{\varepsilon}}{s/\sqrt{N}} > 1.96 \\ or \\ U = \dfrac{\bar{\varepsilon}}{s/\sqrt{N}} < -1.96 \end{cases} \quad (21)$$

where N is the size of the validation dataset, and $\bar{\varepsilon}$, s^2 are the estimated mean and variance of the residuals.

For this simulation example, the true standard deviation of the noise σ_0 is known to be 0.2. Thus, the two-tailed statistical hypothesis test was used to determine whether the variance of the population σ^2 obtained with the different algorithms was statistically different to σ_0^2. The null hypothesis $\mathcal{H}_0$ and its alternative $\mathcal{H}_1$ were therefore:

$$\begin{cases} \mathcal{H}_0: & \sigma^2 = \sigma_0^2 \\ \mathcal{H}_1: & \sigma^2 > \sigma_0^2 \\ & \sigma^2 < \sigma_0^2 \end{cases} \qquad (22)$$

and $\mathcal{H}_0$ can be rejected with a risk level of 5% if:

$$\begin{cases} \Gamma = \dfrac{(N-1).s^2}{\sigma_0^2} < \Gamma_{c1} = \chi^2\left(\upsilon, \dfrac{\alpha}{2}\right) \\ \Gamma = \dfrac{(N-1).s^2}{\sigma_0^2} > \Gamma_{c2} = \chi^2\left(\upsilon, 1-\dfrac{\alpha}{2}\right) \end{cases} \qquad (23)$$

where N is the size of the validation dataset, υ is the number of degrees of freedom, and α is the confidence interval. For this example, the two bounds Γ_{c1} and Γ_{c2} were 439.0 and 562.8. The results of this statistical hypothesis test on the validation datasets are presented in Table 4. This shows that all algorithms gave equivalent results and correctly learned the system. The mean of the residuals is statistically null, and the variance is statistically equal to σ_0^2 for all algorithms.

However, N2PS was not able to prune any parameters, and consistently preserved the initial structure. The best structure obtained with the VNP algorithm includes 49 parameters. The other algorithms gave very similar results. N2PFA and two proposed variants (NNSP-HI-mad and NNSP-IH-mad) found the minimal number of hidden neurons. OBS produced the smallest structure, but required a significant computation time.

Table 4: Best results obtained on the static model dataset

	Structure of NN				RMSE		Statistical Testing (valid.)	
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	U	Γ
without pruning	5	8	57	-	0.166	0.221	0.24	535.2
NNSP-HI-mean	4	4	13	0.69s	0.181	0.200	0.53	500.7
NNSP-HI-mad	5	2	15	0.26s	0.181	0.200	0.24	535.3
NNSP-IH-mean	4	4	15	0.63s	0.181	0.200	0.44	500.1
NNSP-IH-mad	5	2	15	0.23s	0.193	0.206	-0.28	499.2
N2PFA	4	2	13	0.59s	0.181	0.200	0.57	498.0
N2PS	5	8	57	0.04s	0.166	0.221	0.24	535.2
OBS	4	3	9	6.30s	0.181	0.201	0.57	502.5
VNP	4	8	49	0.01s	0.171	0.209	0.63	498.4

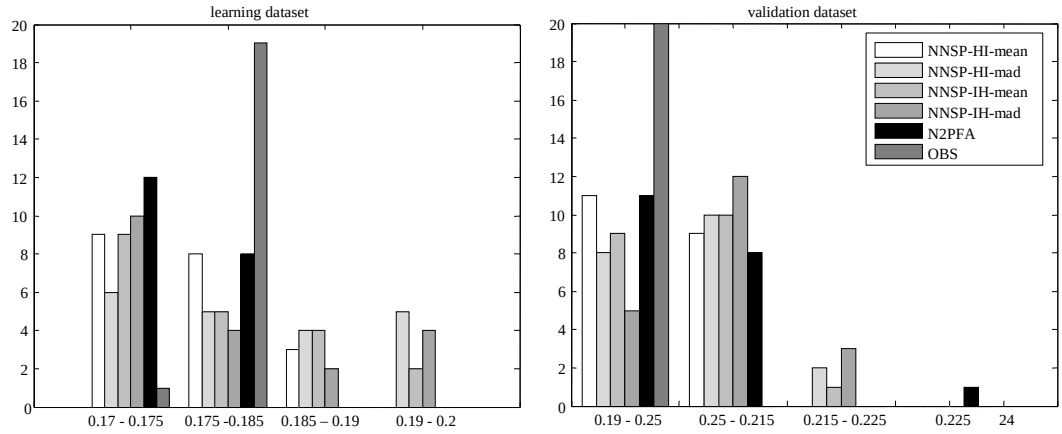

Fig. 4 RMSE distribution on learning and validation datasets

Figure 4 shows the RMSE distribution of the six best-performing algorithms on the learning and validation datasets for the 20 initial sets of weights. This figure shows that these six algorithms consistently found satisfactory results.

6.2. Modeling of a dynamic system

The second system model is also based on a single-hidden-layer perceptron, but uses delayed inputs [89]. This system is described by:

$$y(t) = 1 + \tanh\left(x_1(t-2) - x_2(t) + 3.x_2(t-1)\right) + \tanh\left(x_1(t-2) - x_2(t-2)\right) + e(t) \quad (24)$$

where *e(t)* is additive Gaussian noise with mean 0 and standard deviation 0.17. The delayed inputs x_1 and x_2 are sequences of steps of random length and amplitude. The duration of the steps of input x_1 (respectively x_2) was randomly chosen between 5 and 10 (respectively 8 and 15). The amplitude of x_1 (respectively x_2) was randomly chosen between −1 and 1 (respectively 0 and 1.5). Two datasets of 500 points were created for learning and validation. The input vector used for the learning consisted of the two inputs x_1 and x_2 and their respective delays *t*, *t*–1, *t*–2, *t*–3, and *t*–4. This leads to 10 input neurons for the initial structure of the neural network. The initial learning was performed with 10 hidden neurons.

For all algorithms, two statistical hypothesis tests (21) and (23) were performed to test the mean and variance of the residuals. As in the preceding example, the true standard deviation σ_0 of the noise is known to be 0.17. The two bounds Γ_{c1} and Γ_{c2} are 439.0 and 562.8. Table 5 lists the best results obtained with the eight pruning algorithms and for the learning without pruning. This shows that VNP is not able to find a suitable structure because the mean and variance of the residuals are different from 0 and σ_0^2, respectively. For all other algorithms, the results are equivalent.

Table 5: Best results obtained on the dynamic model dataset

	Structure of NN				RMSE		Statistical Testing (valid.)	
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	U	Γ
without pruning	10	10	121	-	0.148	0.183	1.32	547.1
NNSP-HI-mean	6	5	27	0.7s	0.151	0.174	0.33	522.7
NNSP-HI-mad	10	6	73	0.28s	0.147	0.177	1.21	540.4
NNSP-IH-mean	4	4	25	0.6s	0.154	0.176	0.32	522.8
NNSP-IH-mad	10	2	25	0.55s	0.154	0.175	0.54	526.9
N2PFA	6	8	65	0.93s	0.150	0.180	0.85	521.9
N2PS	5	3	22	0.1s	0.168	0.182	1.32	547.2
OBS	5	4	13	27.32s	0.154	0.172	0.27	520.6
VNP	3	5	26	0.02s	0.235	0.236	5.25	910.9

It can be seen that N2PS was able to prune parameters in only one case. In the 19 other cases, this algorithm preserved the initial structure. OBS determined the best structure, but was again very computationally expensive. N2PFA and the four proposed algorithms produced very similar results, but the four proposed algorithms are less time-consuming. NNSP-IH-mad and NNSP-HI-mad, which use the variance estimation in (12), were unable to prune the inputs. As for the preceding cases, the use of a robust criterion during learning allows the overfitting problem to be avoided.

6.3. Ailerons dataset

The regression problem considered here is also available from WEKA [86]. This dataset addresses a control problem, namely flying an F16 aircraft. The attributes describe the status of the airplane, and the goal is to predict the control action on the ailerons of the aircraft, which comprises 13750 patterns. There are 39 inputs for each pattern. This dataset was randomly divided into two for learning (6868 data) and validation (6882 data). The initial learning was performed with 20 hidden neurons. Because of its computational time requirements, the OBS algorithm was executed only once.

For all algorithms, the two statistical hypothesis tests (21) and (23) were performed to test the mean and variance of the residuals. In this real example, the true variance of the noise σ_0^2 is unknown. In test (23), its value is replaced by the lower variance value obtained on the validation dataset. For this example, the two bounds Γ_{c1} and Γ_{c2} are 6690 and 7076. Table 6 presents the results obtained for the different algorithms. This shows that the VNP algorithm could not find a suitable structure, instead pruning all parameters and becoming trapped in a very bad local minimum. N2PS and one variant of the proposed algorithm (NNSP-IH-mean) were unable to prune any parameters, and therefore preserved the initial structure. These two algorithms and OBS gave results that were statistically worse than those of N2PFA, which produced the best results. OBS again required a considerable computation time (more than one day). The results obtained with the other algorithms are statistically equivalent. N2PFA was the only algorithm that was able to prune the inputs. However, NNSP-HI-mean and NNSP-IH-mad pruned more hidden neurons and parameters, and determined equivalent structures quicker than N2PFA. The computational time for N2PFA is four times higher than those for the proposed algorithms.

Table 6: Best results obtained on the ailerons dataset

	Structure of NN				RMSE		Statistical Testing (valid.)	
	# of inputs	# of hidden	# of parameters	Duration	Training	Validation	U	Γ
without pruning	39	20	821	-	$1.61*10^{-4}$	$1.65*10^{-4}$	0.89	7106
NNSP-HI-mean	39	4	165	3min 34s	$1.57*10^{-4}$	$1.63*10^{-4}$	0.27	6908
NNSP-HI-mad	39	15	616	2min 6s	$1.58*10^{-4}$	$1.63*10^{-4}$	0.41	6914
NNSP-IH-mean	39	20	821	20.93s	$1.61*10^{-4}$	$1.65*10^{-4}$	0.89	7106
NNSP-IH-mad	39	7	288	4min 23s	$1.58*10^{-4}$	$1.63*10^{-4}$	0.89	6880
N2PFA	23	9	226	16min 59s	$1.58*10^{-4}$	$1.62*10^{-4}$	-1.11	6882
N2PS	39	20	821	9.75s	$1.61*10^{-4}$	$1.65*10^{-4}$	0.89	7106
OBS	38	20	601	27h 19min 25s	$1.64*10^{-4}$	$1.68*10^{-4}$	0.20	7331
VNP	0	0	1	0.312s	$29.66*10^{-4}$	$29.67*10^{-4}$	-17.05	43079

7 Conclusions

In this paper, a new pruning algorithm was proposed to determine the optimal structure of a MLP for both classification and regression problems. Four variants of this algorithm were tested and compared with four classical pruning algorithms on three classification problems and three regression problems. The proposed algorithms produced equivalent results or outperformed the four comparative algorithms. Moreover, they required less computational time than the OBS and N2PFA algorithms.

Of the four proposed variants, NNSP-HI-mean gave the best results. This variant begins by pruning hidden neurons, then considers the inputs, and finally the parameters. It uses the variance estimator (10) proposed by Engelbrecht [26]. The results have shown that the use of a robust learning algorithm allows the overfitting problem to be avoided in both classification and regression problems.

References

1. Rumelhart DE, McClelland JL (1986) Parallel distributed processing: Exploration in the microstructure of cognition Vol.1. MIT Press, Cambridge
2. Han HG, Qiao JF (2013) A structure optimisation algorithm for feedforward neural network construction. Neurocomputing 99: 347-357
3. Patel MC, Panchal M (2012) A review on ensemble of diverse artificial neural networks. Int. J. of Advanced Research in Computer Engineering and Technology 1: 63-70
4. Drucker H (2002) Effect of pruning and early stopping on performance of a boosting ensemble. Computationnal Statistics and Data Analysis 393-406
5. Kerlirzin P, Vallet F (1993) Robustness in multilayer perceptrons. Neural Computation 5: 473-482
6. Thomas P, Bloch G, Sirou F, Eustache V (1999) Neural modeling of an induction furnace using robust learning criteria. Journal of Integrated Computer Aided Engineering 6: 15-25
7. Williams PM (1995) Bayesian regularization and pruning using a Laplace prior. Neural Computation 7:117-143
8. Bartlett PL (1997) For valid generalization, the size of the weights is more important than the size of the network. Advances in Neural Information Processing Systems 9:134-140. http://yaroslavvb.com/papers/bartlett-for.pdf. Accessed 10 July 2013
9. Cawley GC, Talbot NLC (2007) Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. Journal of Machine Learning Research 8: 841–861
10. Bishop CM (1995) Neural networks for pattern recognition. Clarendon Press, Oxford
11. Beigy H, Meybodi MR (2009) A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks. International Journal of Systems Science 40: 101-118
12. Kruschke JH (1989) Improving generalization in backpropagation networks. Proc. of Int. Joint Conf. on Neural Networks IJCNN'89 1: 443-447
13. LeCun Y, Denker J, Solla S, Howard RE, Jackel LD (1990) Optimal brain damage. In: Touretzky DS (ed) Advances in Neural Information Processing Systems, Morgan Kaufmann, San Mateo
14. Hassibi B, Stork DG (1993) Second order derivatives for network pruning: optimal brain surgeon. In: Hanson SH, Cowan JD, Gilles CL (Eds.) Advances in Neural Information Processing Systems, Morgan Kaufmann, San Mateo

15. Reed R. (1993) Pruning algorithm – A survey. IEEE Transactions on Neural Networks 4: 740–747
16. Jutten C, Fambon O (1995) Pruning methods: a review. Proc. of European Symp. on Artificial Neural Network ESANN'95 129–140
17. Mezard M, Nadal JP (1989) Learning in feedforward neural networks: the tiling algorithm. Journal of Physics 22: 1285-1296
18. Fahlman SE, Lebier C (1990) The cascade-correlation learning architecture. Computer Science Department paper 138. http://repository.cmu.edu/compsci/1938. Accessed 10 July 2013
19. Yeung DY (1991) Automatic determination of network size for supervised learning. IEEE Int. J. Conf. on Neural Networks IJCNN'91 1: 158-164
20. Chentouf R, Jutten C (1996) Combining sigmoids and radial basis function in evolutive neural architecture. European Symp. on Artificial Neural Network ESANN'96 129–134
21. Hirose Y, Yamashita K, Hijita S (1991) Back-propagation algorithm which varies the number of hidden units. Neural Networks 4: 61-66
22. Nabhan TM, Zomaya AY (1994) Toward generating neural network structures for function approximation. Neural Networks 7: 89-99
23. Narasimha PL, Delashmit WH, Manry MT, Li J, Maldonado F (2008) An integrated growing-pruning method for feedforward network training. Neurocomputing 71: 2831-2847
24. Whitley D, Bogart C (1990) The evolution of connectivity: pruning neural networks using genetic algorithms. Proc. of Int. J. Conf. on Neural Networks IJCNN'90 134-137
25. Angeline PJ, Saunders GM, Pollack JB (1994) Evolutionary algorithm that construct recurrent neural networks. IEEE Transactions on Neural Networks 5: 54-65
26. Engelbrecht AP, (2001) A new pruning heuristic based on variance analysis of sensitivity information. IEEE transactions on Neural Networks 12: 1386-1399
27. Kolmogorov AN (1957) On the representational of continuous functions of many variables by superpositions of continuous functional of one variable and addition. Doklady Akademii Nauk URSS 114: 953-956
28. Cybenko G (1989) Approximation by superposition of a sigmoïdal function. Math. Control Systems Signals 2: 303-314
29. Funahashi K (1989) On the approximate realisation of continuous mapping by neural networks. Neural Networks 2: 183-192
30. Hecht-Nielsen R (1987) Kolmogorov's mapping neural network existence theorem. IEEE First Annual Int. Conf. on Neural Networks 3: 11-13
31. Huang GB (2003) Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Transactions on Neural Networks 14: 274-281
32. Dreyfus G, Martinez JM, Samuelides M, Gordon MB, Badran F, Thiria S, Hérault L (2002) Réseaux de neurones: méthodologies et applications. Editions Eyrolles, Paris
33. Yao X (1993) Evolutionary artificial neural networks. International Journal of Neural Systems 4: 203-222

34. Huang DS, Du JX (2008) A constructive hybrid structure approach to solve routing problems. IEEE Transactions on Neural Networks 19: 2099-2115
35. Mantzaris D, Anastassopoulos G, Adamopoulos A (2011) Genetic algorithm pruning of probabilitic neural networks in medical disease estimation. Neural Networks 24: 831-835
36. Stathakis D (2009) How many hidden layers and nodes? International Journal of Remote Sensing 30: 2133-2147
37. Yang SH, Chen YP (2012) An evolutionary constructive and pruning algorithm for artificial neural networks and its applications. Neurocomputing 86: 140-149
38. Maniezzo V (1994) Genetic evolution of the topology and weight distribution of neural networks. IEEE Transactions on Neural Networks 5: 39-53
39. Wilamowski BM, Cotton NJ, Kaynak O, Dundar G (2007) Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks. Proc. of IEEE ISIE 3298-3789
40. Puma-Villanueva WJ, Von Zuben FJ (2010) Evolving arbitrarily connected feedforward neural networks via genetic algorithm. Proc. of the Brazilian Symp. on Artificial Neural Networks 23-28
41. Puma-Villanueva WJ, Dos Santos EP, Von Zuben FJ (2012) A constructive algorithm to synthesize arbitrarily connected feedforward neural networks. Neurocomputing 75: 23-28
42. Kwok TY, Yeung DY (1997) Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Transactions on Neural Networks 8: 630-645
43. Ash T, (1989) Dynamic node creation in backpropagation networks. Connection Science 1: 365-375
44. Setiono R, Hui LCK (1995) Use of a quasi-Newton method in a feedforward neural network construction algorithm. IEEE Transactions on Neural Networks 6: 273-277
45. Lee TC (1991) Neural network systems techniques and applications: Algorithms and architectures. Academic Press, New-York
46. Weng W, Khorasani K (1996) An adaptive structural neural network with application to EEG automatic seizure detection. Neural Networks 9: 1223-1240
47. Prechelt L (1997) Investigation of the CasCor family of learning algorithms. Neural Networks 10: 885-896
48. Ma L, Khorasani K (2004) New training strategies for constructive neural networks with application to regression problems. Neural Networks 17: 589–609
49. Ma L, Khorasani K (2005) Constructive feedforward neural networks using Hermite polynomial activation functions. IEEE Transactions on Neural Networks 16: 821-833
50. Islam M, Sattar A, Amin F, Yao X, Murase K (2009) A new constructive algorithm for architectural and functional adaptation of artificial neural networks. IEEE Transactions on Systems Man and Cybernetics part B 39: 1590-1605
51. Bebis G, Georgiopoulos M (1994) Feed-forward neural networks: why network size is so important. IEEE Potentials 13: 27-31
52. Khosravi A, Nahavendi S, Creighton D (2010) A prediction interval-based approach to determine optimal structures of neural network metamodels. Expert Systems with Application 37: 2377-2387

53. Chauvin Y (1988) A back-propagation algorithm with optimal use of hidden units. In Touretzky DS (ed) Advances in Neural Information Processing 519-526. http://books.nips.cc/papers/files/nips01/0519.pdf. Accessed 10 July 2013
54. Weigend AS, Rumelhart DE, Huberman BA (1990) Generalization by weight-elimination with application to forecasting. In Lippmann R, Moody J Touretzky DS (eds) Advances in Neural Information Processing 875-882, http://books.nips.cc/papers/files/nips03/0875.pdf. Accessed 10 July 2013
55. Nowland SJ, Hinton GE (1992) Simplifying neural networks by soft weight-sharing. Neural Computation 4: 473-493
56. Larsen J, Hansen LK (1994) Generalization performance of regularized neural network models. IEEE Workshop on Neural Network for Signal Processing, 42-51
57. Leung CS, Wang HJ, Sum J (2010) On the selection of weight decay parameter for faulty networks. IEEE Transactions on Neural Networks 21: 1232-1244
58. Wang J, Wu W, Zurada JM (2012) Computational properties and convergence analysis of BPNN for cyclic and almost cyclic learning with penalty. Neural Networks 33: 127-135
59. Setiono R, Leow WK (2000) Pruned neural networks for regression. 6th Pacific RIM Int. Conf. on Artificial Intelligence PRICAI'00 500-509
60. Norgaard M (1996) System identification and control with neural networks, Ph.D. dissertation, Institute of Automation, Technical University of Denmark
61. Thomas P, Bloch G (1998) Robust pruning for multilayer perceptrons. IMACS/IEEE Multiconference on Computational Engineering in Systems Applications CESA'98 17–22, 1998
62. Leung CS, Wang HJ, Sum J, Chan LW (2001) A pruning method for the recursive least squared algorithm. Neural Networks 14: 147-174
63. Tang HS, Xue ST, Chen R, Sato T (2007) H∞ Filtering method for neural network training and pruning. J. Comp. in Civ. Engineering 21: 47–58
64. Ricotti ME, Zio E (1999) Neural network approach to sensitivity and uncertainty analysis. Reliability Engineering and System Safety 64: 59–71
65. Zeng X, Yeung DS (2006) Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure. Neurocomputing 69: 825–837
66. Chandrasekaran H, Chen HH, Maury MT (2000) Pruning of basis functions in nonlinear approximators. Neurocomputing 34: 29–53
67. Lauret P, Fock E, Mara TA (2006) A node pruning algorithm based on a Fourier amplitude sensitivity test method. IEEE Transactions on Neural Networks 17: 273–293
68. Augasta MG, Kathirvalavakumar T (2011) A Novel Pruning Algorithm for Optimizing Feedforward Neural Network of Classification Problems. Neural Processing Letters 34: 241-258
69. Levin AU, Leen TK, Moody JE (1994) Fast pruning using principal components. In Cowan J, Tesauro G, Alspector J (eds) Advances in Neural Information Processing. http://books.nips.cc/papers/files/nips06/0035.pdf. Accessed 10 July 2013.

70. Medeiros CMS, Baretto GA (2013) A novel weight pruning method for MLP classifiers on the MAXCORE principle. Neural Computation and Applications 22: 71-84
71. Liang X (2007) Removal of hidden neurons in MLP by orthogonal projection and weight crosswise propagation. Neural Computing and Applications 16: 57–68
72. Rivals I, Personnaz L (2003) Neural network construction and selection in nonlinear modeling. IEEE Transactions on Neural Networks 14: 804–819
73. Huang GB, Saratchandran P, Sundararajan N (2005) A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Transactions on Neural Networks 16: 57-67
74. Hsu CF (2008) Adaptive growing and pruning neural network control for a linear piezoelectric ceramic motor. Engineering Applications of Artificial Intelligence 21: 1153-1163
75. Qi M, Zhang GP (2001) An investigation of model selection criteria for neural network time series forecasting. European Journal of Operational Research 132: 666-680
76. Egrioglu E, Aladag CH, Gunay S (2008) A new model selection strategy in artificial neural networks. Applied Mathematics and Computation 195: 591-597
77. Hand D, Mannila H, Smyth P (2001) Principles of data mining. The MIT press, Cambridge
78. Thomas P, Bloch G (1997) Initialization of one hidden layer feed-forward neural networks for non-linear system identification. Proc. of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics WC'97 295–300
79. Nguyen D, Widrow B (1990) Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. Proc. of the Int. J. Conf. on Neural Networks IJCNN'90 3: 21-26
80. Demuth H, Beale P (1994) Neural networks toolbox user's guide V2.0. The MathWorks, Inc.
81. Huber PJ (1964) Robust estimation of a location parameter. Ann. Math. Stat. 35: 73-101
82. Ruck DW, Rogers SK, Kabrisky M (1990) Feature selection using a multilayer perceptron. Neural Network Computing 2: 40–48
83. Tarr G (1991) Multilayered feedforward networks for image segmentation, Ph.D. dissertation, Air Force Inst. Technol. Wright-Patterson AFB
84. Ljung L (1987) System identification: theory for the user. Prentice-Hall, Englewood Cliffs
85. Smith JW, Everhart JE, Dickson WC, Knowler WC, Johannes RS (1988) Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. Proc. of the Symp. on Computer Application and Medical Care 261-265
86. WEKA. http://weka.wikispaces.com/Datasets. Accessed 10 July 2013
87. Sigillito VG, Wing SP, Hutton LV, Baker KB (1989) Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest 10: 262-266
88. Noyel M, Thomas P, Charpentier P, Thomas A, Beaupretre B (2013) Improving production process performance thanks to neuronal analysis. Proc. of 11th IFAC Workshop on Intelligent Manufacturing System IMS'13

89. Thomas P, Thomas A (2008) Elagage d'un perceptron multicouches : utilisation de l'analyse de la variance de la sensibilité des paramètres. 5ème Conférence Internationale Francophone d'Automatique CIFA'08