Combined input variable selection and model complexity control for nonlinear regression

Timo Similä [a,b], Jarkko Tikka [a,*]

[a] *Laboratory of Computer and Information Science, Helsinki University of Technology, P.O. Box 5400, FIN-02015 HUT, Finland*

[b] *Xtract Ltd, Hitsaajankatu 22, FIN-00810 Helsinki, Finland*

Abstract

Choosing a useful combination of input variables and an appropriate complexity of the model is an essential task in nonlinear regression analysis because of the risk of overfitting. This article provides a workable solution for the multilayer perceptron model. An initial structure of the model, including all the input variables, is fixed in the beginning. Only the most useful input variables and hidden nodes remain effective when the model is fitted with the proposed penalization method. The method is tested on three benchmark data sets. Experimental results show that the removal of useless input variables and hidden nodes from the model improves its generalization capability. In addition, the proposed method compares favorably with respect to other penalization methods.

Key words: Regression, Function approximation, MLP, Multilayer perceptron, Feedforward network, Input variable selection, Hidden node selection

1 Introduction

Regression is the problem of learning an input-output relationship based on data. The functional form of the relationship is unknown in general. The multilayer perceptron is a convenient choice of model, because it is capable of approximating a wide range of nonlinear functions without restrictive assumptions (Bishop, 1995; White, 1990). The quality of the approximation depends on the complexity of the model. It can be controlled by varying the number of

* Corresponding author. Tel.: +358 9 451 3543; fax: +358 9 451 3277. E-mail address: tikka@cis.hut.fi

hidden nodes and trying different combinations of input variables, or by penalizing the weights between the units of the network in some sense (Anders and Korn, 1999; Gallinari and Cibas, 1999).

The amount of collected data has increased substantially in many applications and this trend is likely to continue in the future. As a consequence, a lot of input variables are typically available for learning the input-output relationship. Input selection in regression means identifying the input variables that are useful in predicting the values of the output variables. Three objectives motivate input selection (Guyon and Elisseeff, 2003). First, generalization capability may be increased. It is often unwise to include all the available input variables in the multilayer perceptron, because useless variables increase the likelihood of overfitting (Zhang, 2007). Second, cost-effective models are obtained. It is faster to fit the models and if measuring of the variables is expensive economic savings are achieved. Third, understanding of the underlying process that has generated the data is improved, since useless variables are dropped from the model.

The input selection methods can be divided into three classes, which are filter, wrapper, and embedded approaches (Blum and Langley, 1997; Guyon and Elisseeff, 2003; Kohavi and John, 1997). In the filter approach, the input selection procedure is independent from the fitting of the final prediction model, which is typically nonlinear (here the multilayer perceptron). In the first phase, a subset of input variables is identified using, for example, techniques that exist for linear models (Bi et al., 2003; Tikka et al., 2006) or linear-in-the-parameter models (Li and Peng, 2007). A model-free technique, such as mutual information maximization (François et al., 2007), is another possibility. In the second phase, the final prediction model is fitted using the selected input variables. The filter approach is computationally tractable, since the input selection process is fast and the final prediction model is fitted only once. On the other hand, the subset of input variables that is found to be optimal in the first phase may not be optimal in the second phase.

In the wrapper approach, input variable subsets are ranked according to some estimate of generalization capability of the model (Kohavi and John, 1997). Cross-validation error is an often used estimate. The variable selection process is supported by a general-purpose search algorithm that proposes promising combinations. Forward selection and backward elimination are the most typical search algorithms. The performance of the final prediction model is optimized directly, which is advantageous over the two-phase filter approach. However, the wrapper approach is computationally more complex, because the potentially time-consuming model fitting has to be performed several times.

In the embedded approach, input selection is incorporated as a part of the fitting process. It is highly specific to the model structure compared with the

more universal wrapper approach, where the model is treated as a black box. In the context of multilayer perceptrons, variable selection can be seen as a special case of network pruning (Castellano and Fanelli, 2000; Hassibi and Stork, 1993; Le Cun et al., 1990). It provides a sequence of simplified networks without refitting from scratch for every variable subset investigated. Certain penalization techniques offer another more direct category of embedded methods. The model fitting is penalized in such a way that the parameters associated with some input variables become zero during the optimization process. Such techniques have been proposed for multilayer perceptrons (Chapados and Bengio, 2001; Grandvalet and Canu, 1999) and radial basis function networks (Tikka, 2007). Estimated generalization capability, for example, can be used to determine the stopping condition in network pruning and the optimal amount of penalization in direct methods. The embedded approach is usually computationally much lighter than the wrapper approach.

In this article, a simultaneous input variable and hidden node selection technique for the multilayer perceptron is proposed. The technique belongs to the category of direct embedded methods. The proposed cost function bears similarity to the well-known weight decay penalization (Bishop, 1995). It is, however, conceptually closer to input decay (Chapados and Bengio, 2001), which aims at forcing all the outcoming weights of useless input variables together close to zero. The proposed penalty function differs from weight decay and input decay in being capable of producing groups of exactly zero weights. It can, thereby, remove some input variables and hidden nodes completely from the network. The theory that supports this property is originally derived in the context of a multiple response linear regression model (Similä, 2007; Similä and Tikka, 2007). Such a model supplemented with a nonlinear activation function constitutes one layer of the network, which makes the theory applicable to the multilayer perceptron as well. The maximum number of hidden nodes is given in advance, but only some of them are active in the network after the optimization process. Existing approaches to simultaneous input variable and hidden node selection operate by adding or deleting units of the network one-by-one (Steppe et al., 1996; Rivals and Personnaz, 2003; Yacoub and Bennani, 2000). In contrast, the proposed method is defined as an optimization problem of continuous-valued parameters.

The rest of the article is organized as follows. Section 2 starts by a short introduction to the multilayer perceptron. It is followed by a presentation of the proposed cost function and detailed analysis of its optimization. The results of experiments on three benchmark data sets are given in Section 3. Section 4 concludes the article.

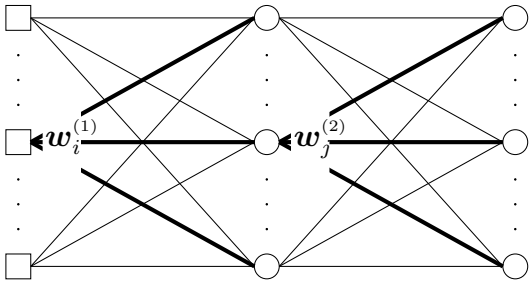

Fig. 1. The architecture of a one-hidden-layer network. Thick lines represent the outcoming weights of the ith input variable (left) and the jth hidden node (right). The bias terms $\boldsymbol{w}_0^{(1)}$ and $\boldsymbol{w}_0^{(2)}$ are not visible. The proposed penalization technique aims at forcing all the outcoming weights of useless units together to zero, so some input variables and hidden nodes are completely removed from the network.

2 Methods

2.1 Multilayer Perceptron

The multilayer perceptron is a standard method for representing nonlinear mappings between input variables $\boldsymbol{x}$ and output variables $\boldsymbol{y}$. It is a network of successive layers that aim at transforming the input variables so that the resulting model $\boldsymbol{f}(\boldsymbol{x})$ would predict the output variables accurately. Fig. 1 depicts the architecture of a multilayer perceptron. The first layer consists of linear combinations of the input variables which are transformed by an activation function. The successive (hidden) layers operate in the same way with the only difference that they process the outcoming signals of the previous layer. The model $\boldsymbol{f}(\boldsymbol{x})$ is parameterized by the weights of the linear combinations $\boldsymbol{w}$. A mathematical description of the multilayer perceptron is presented in Appendix A and a complete discussion is given by Bishop (1995).

There are several free parameters in the architecture of a multilayer perceptron. The types of activation functions and the number of hidden layers must be determined. A standard choice, which is also adopted in this article, is to use the hyperbolic tangent activation function. In the case of a regression task, however, the activation functions are linear in the last layer. It can be shown that such networks with only a single hidden layer can approximate arbitrarily well any functional continuous mapping from one finite-dimensional space to another, provided that the number of hidden nodes is sufficiently large (Hornik et al., 1989, 1990). Despite this beautiful theoretical property, it is very difficult to give a general rule for the sufficient number of hidden nodes in practice. Another difficulty is to determine the combination of input variables that is useful in the prediction task. In the remaining of this section, a framework for combined input variable and hidden node selection is introduced.

Overfitting is an inherent trouble in nonlinear regression. It means that the model is very accurate on the training set, but it has poor accuracy on previously unseen samples of data. To tackle overfitting, it is common to build a complex enough network and then constrain its flexibility somehow. Such an approach leads to examining the problem

$$\text{minimize } E(\boldsymbol{w}) + \lambda P(\boldsymbol{w}), \tag{1}$$

where the parameter $\lambda \geq 0$ controls the tradeoff between the fit on the training set (maximal at $\lambda = 0$) and the simplicity of the network (maximal when λ is large). The former aspect is measured by the mean squared error function [1]

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{y}_n - \boldsymbol{f}(\boldsymbol{x}_n)\|^2, \tag{2}$$

where the N samples in the training set are indexed by n. The latter aspect is measured by the penalty function

$$P(\boldsymbol{w}) = \sum_{\ell=1}^{L} \sum_{i=1}^{I(\ell)} p(\|\boldsymbol{w}_i^{(\ell)}\|), \tag{3}$$

where it is assumed that $\boldsymbol{x} \in \mathbb{R}^{I(1)}$ and there are $I(\ell)$ nodes in the hidden layer $\ell \in \{2, \ldots, L\}$. The expression $\boldsymbol{w}_i^{(\ell)}$ denotes the outcoming weights of the ith unit in the ℓth layer. Note that the bias terms $\boldsymbol{w}_0^{(\ell)}$ are not present in Eq. (3), so they are free of penalization. The value of λ must be fixed by cross-validation or related methods so that error function (2) is as minimal as possible for novel data.

The choice $p(s) = s^2$ leads to weight decay, which is a well-known penalization technique (Bishop, 1995). It prevents the weights from being far from zero, but it does not generally force them to be exactly zero. All the weights are penalized in the same way, since penalty function (3) is simply the sum of squares. Thereby, the weight decay penalization does not change the architecture of the network in any sense.

Taking any other $p(s)$ that is increasing on $s \geq 0$, the outcoming weights are penalized in groups, as illustrated in Fig. 1. In the context of multilayer perceptrons, grouped penalization techniques are discussed by Chapados and Bengio (2001); Grandvalet and Canu (1999). Particularly, the choice $p(s) = s^2/(\eta + s^2)$ with $\eta > 0$ leads to input decay (Chapados and Bengio, 2001).

[1] Throughout the article, $\|\cdot\|$ denotes the L_2-vector-norm defined as the square root of the component sum of squares.

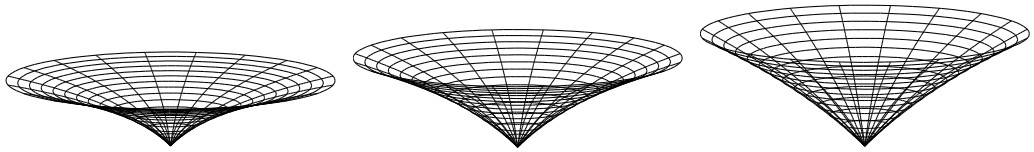

Fig. 2. The mapping $p(\|\boldsymbol{w}_i^{(\ell)}\|)$ in the bivariate case using the function $p(s) = c\log(1 + s/c)$. The value of $c > 0$ is doubled in each subfigure from left to right. The degree of concavity decreases in the same direction. Note the nondifferentiability at the origin, which is the key to sparsity.

It does not either change the architecture in a strict sense, because $p(s)$ is nearly quadratic for small nonnegative values of s. Actually, with a large η, input decay resembles weight decay. As η approaches zero, $p(s)$ approaches the step function. This finally encourages sparsity, but at the same time the optimization becomes more difficult. Chapados and Bengio (2001) only present the penalization of outcoming weights of the input variables, but there should be no restraints on covering the hidden nodes as well.

In order to discard unnecessary units of the network, the corresponding block norms $\|\boldsymbol{w}_i^{(\ell)}\|$ must be exactly zero. Assuming that $p(s)$ is differentiable on $s \geq 0$, penalty function (3) encourages blockwise sparse solutions to problem (1) only when $p'(0) > 0$ holds, as it has been shown by Similä (2007). For example, the Group Lasso penalization (Bakin, 1999; Yuan and Lin, 2006) is linear with $p'(s) \equiv 1$ and it satisfies this condition. On the other hand, both input decay and weight decay have $p'(0) = 0$ and they fail to produce sparse solutions. A disadvantage of the sparsity condition $p'(0) > 0$ is that penalty function (3) becomes nondifferentiable at certain points, so problem (1) cannot simply be solved with gradient-based methods.

In this article, it is proposed to use the function

$$p(s) = c\log(1 + s/c) \tag{4}$$

with $c > 0$ to perform simultaneous input selection and node pruning. The sparsity condition holds, because we have $p'(0) = 1$. Fig. 2 illustrates function (4). The strict concavity of $p(s)$ on $s \geq 0$ is important, since it reduces the unnecessary modeling bias when the true unknown value of $\|\boldsymbol{w}_i^{(\ell)}\|$ is large (Fan and Li, 2001). The parameter c controls the degree of concavity, which is the strongest with small positive values. In the limit, as $c \to \infty$, function (4) approaches the Group Lasso function $p(s) = s$. The main contributions of this article are penalty function (4) and an efficient optimization method that is introduced next.

2.3 Optimization

The mapping $p(\|\boldsymbol{w}_i^{(\ell)}\|)$ is nondifferentiable at the origin when the proposed function $p(s)$, as defined in Eq. (4) and illustrated in Fig. 2, is applied. Problem (1) cannot be solved directly using gradient-based methods for $\lambda > 0$ in that case. To overcome the difficulty, a slightly modified problem

$$\begin{aligned} &\text{minimize } E(\boldsymbol{w}) + \lambda \sum_{\ell=1}^{L} \sum_{i=1}^{I(\ell)} p(s_i^{(\ell)}) \\ &\text{subject to } s_i^{(\ell)} \geq \|\boldsymbol{w}_i^{(\ell)}\| \end{aligned} \tag{5}$$

is considered. Here, $s_i^{(\ell)}$ denotes a slack variable. Since $p(s)$ is increasing, the constraints must hold with equality at the solution. This notice highlights the close connection between problems (1) and (5). The latter one is, however, significantly easier to solve, because it consists of a differentiable objective and convex cone constraints.

Problem (5) can be approximated by the unconstrained minimization of the logarithmic barrier reformulation

$$Q(\boldsymbol{w}, \boldsymbol{s}) = E(\boldsymbol{w}) + \lambda \sum_{\ell=1}^{L} \sum_{i=1}^{I(\ell)} p(s_i^{(\ell)}) - \mu \sum_{\ell=1}^{L} \sum_{i=1}^{I(\ell)} \log((s_i^{(\ell)})^2 - \|\boldsymbol{w}_i^{(\ell)}\|^2). \tag{6}$$

The parameter $\mu > 0$ determines the quality of the approximation such that small positive values correspond to the highest accuracy. Function (6) is differentiable, because the cone constraints are squared on both sides before feeding into the logarithmic barrier function. More details about the barrier approach to inequality constrained optimization are given by Boyd and Vanderberghe (2004). Any gradient-based method could be applied to minimize function (6) with respect to $\boldsymbol{w}$ and $\boldsymbol{s}$. However, as will be explained shortly, it is possible to get rid of the slack variables completely so that the gradients need to be taken with respect to $\boldsymbol{w}$ only.

Function (6) is separable in the slack variables, so its partial derivative with respect to one slack variable

$$\frac{\partial Q(\boldsymbol{w}, \boldsymbol{s})}{\partial s_i^{(\ell)}} = \frac{\lambda c}{c + s_i^{(\ell)}} - \frac{2\mu s_i^{(\ell)}}{(s_i^{(\ell)})^2 - \|\boldsymbol{w}_i^{(\ell)}\|^2} \tag{7}$$

does not depend on the others. For a fixed $\boldsymbol{w}$, the stationary points of function (6) can be found by requiring derivative (7) to be zero for all the slacks. This corresponds to computing the roots of quadratic equations. Assuming

$0 < \mu < \frac{1}{2}\lambda c$, the only zero point of derivative (7) on $s_i^{(\ell)} > \|\boldsymbol{w}_i^{(\ell)}\|$ is

$$\hat{s}_i^{(\ell)} = \frac{\mu + \sqrt{\mu^2 + (\lambda - 2\mu/c)\lambda \|\boldsymbol{w}_i^{(\ell)}\|^2}}{\lambda - 2\mu/c}, \tag{8}$$

because $\lambda > 0$ and $c > 0$. It is straightforward to compute the second-order partial derivative of function (6) with respect to $s_i^{(\ell)}$ and see that it is positive at $\hat{s}_i^{(\ell)}$. As an important consequence, treating $\boldsymbol{w}$ as a constant, Eq. (8) gives the unique minimum point of function (6) on the feasible domain [2] in a closed form.

In order to minimize function (6) with respect to both $\boldsymbol{w}$ and $\boldsymbol{s}$, it suffices to minimize function $\hat{Q}(\boldsymbol{w}) = Q(\boldsymbol{w}, \hat{\boldsymbol{s}}(\boldsymbol{w}))$ with respect to $\boldsymbol{w}$, where the slack variables depend on $\boldsymbol{w}$ through Eq. (8). Among a wide variety of choices, the scaled conjugate gradient algorithm (Møller, 1993) is used in the experiments in Section 3. The algorithm requires the objective function $\hat{Q}(\boldsymbol{w})$ and its partial derivatives

$$\frac{\partial \hat{Q}(\boldsymbol{w})}{\partial \boldsymbol{w}_i^{(\ell)}} = \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}_i^{(\ell)}} + \frac{\lambda c}{\hat{s}_i^{(\ell)}(c + \hat{s}_i^{(\ell)})} \boldsymbol{w}_i^{(\ell)} \tag{9}$$

for $i = 1, \ldots, I(\ell)$ and $\ell = 1, \ldots, L$. The partial derivative of $\hat{Q}(\boldsymbol{w})$ with respect to the bias term $\boldsymbol{w}_0^{(\ell)}$ is simply $\frac{\partial}{\partial \boldsymbol{w}_0^{(\ell)}} E(\boldsymbol{w})$. The value of μ is decreased logarithmically in fifteen steps from $\frac{1}{3}\lambda c$ to 10^{-8} in the course of iteration. One hundred iterations are performed for each value of μ, which makes 1500 iterations in total. Each time the value of μ is changed the conjugate direction is reinitialized to the steepest descent direction. The computational complexity of one iteration with the proposed penalty function is practically the same as with weight decay and input decay. Since we fix the number iterations with all the penalty functions to 1500 in the experiments of the next section, there are no differences in computation times.

3 Experiments

The performance of multilayer perceptrons that are trained using the proposed penalty function (see Eqs. (3) and (4)) is illustrated and compared to the input decay and weight decay approaches. The same type of penalization is applied to the outcoming weights of all units of a network. This differs from the original input decay (Chapados and Bengio, 2001), where only the outcoming weights

[2] There are no explicit constraints in the minimization of function (6), but the logarithm is only defined for positive values.

of input variables are penalized. Three benchmark data sets are used, which can all be downloaded from the Delve data repository [3].

The first data set is called Pumadyn and it is from a realistic simulation of the dynamics of a Puma 560 robot arm. The task is to predict the angular acceleration of one of the arm's links using 32 input variables. They represent angular positions, velocities, and torques of the robot arm. It is known that the dependencies are nonlinear and the amount of noise is high.

The second data set is called Add10 and it has been introduced by Friedman (1991). The data set consists of ten input variables and a single output variable, which is generated from the model

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon, \tag{10}$$

where ε is Gaussian noise with zero mean and unit variance. The input variables are sampled independently from a uniform distribution in the range $x_i \in [0, 1]$. Observe that the output variable does not depend on the last five input variables, which are thereby irrelevant.

The third data set is a collection of computer system activity measures. The computer was used in a multi-user environment. The objective is to predict the portion of time that the central processing unit was in the user mode. There are 21 input variables available for the prediction task.

In each of the three data sets, the first one thousand samples are used as a training set, the following one thousand samples as a validation set, and the rest of the samples as a test set. Over six thousand samples are left to the test set in each case. The values of all variables are scaled to have zero mean and unit variance before the analysis.

The networks are evaluated using 40 values of the penalty parameter λ, which are equally spaced on a logarithmic scale in the ranges $[0.005, 0.5]$, $[10^{-6}, 0.8]$, and $[10^{-5}, 0.5]$, in the cases of Pumadyn, Add10, and Computer data, respectively. The same values of λ are used for all choices of penalization. In the proposed penalty function, the following values are used for the concavity parameter $c \in \{0.1, 0.3, 0.7, 1.5, 4, 100\}$. The value $c = 100$ already approximates the Group Lasso function $p(s) = s$ very closely. Input decay is applied with the parameter values $\eta \in \{0.01, 0.09, 0.3, 0.8, 2, 7\}$. All networks have an input layer and one hidden layer ($L = 2$) and the number of available hidden nodes is $I(2) = 10$. The Nguyen-Widrow method (Nguyen and Widrow, 1990) is applied to initialize the weights of a network. The initialization and the subsequent optimization are repeated five times in order to avoid local minima. The weights resulting in the smallest mean squared error on the training set

[3] http://www.cs.toronto.edu/~delve/data/datasets.html

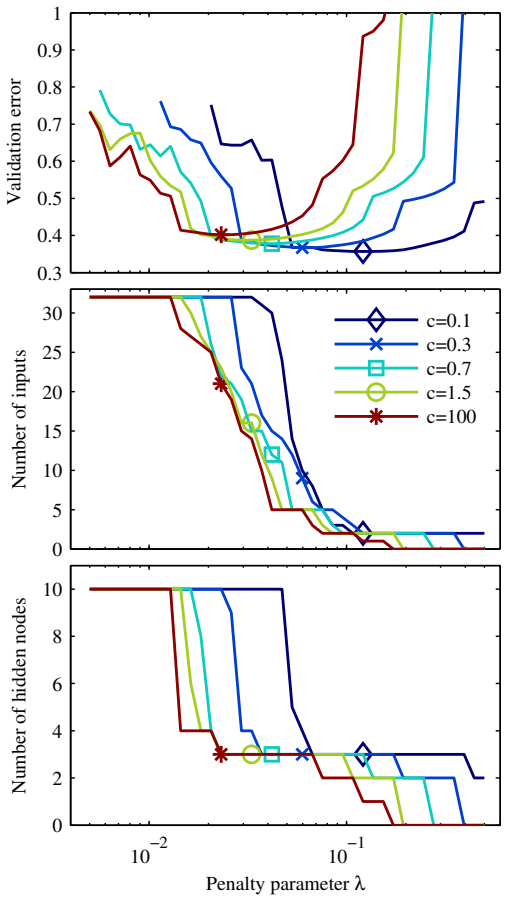

Fig. 3. Mean squared validation error (top), the number of selected input variables (middle), and the number of selected hidden nodes (bottom) for different values of the concavity parameter c as a function of the penalty parameter λ in the case of Pumadyn data. Markers indicate the positions of the minimum mean squared validation errors in each subfigure.

are included in further analysis.

A unit of the network (an input variable or a hidden node) is considered to be useless if the L_∞-norm (the largest entry in absolute values) of its outcoming weights $\boldsymbol{w}_i^{(\ell)}$ is less than 0.01 after the optimization process. The threshold is not a parameter of the penalized least squares approach that is introduced in Section 2, but it is needed in the illustration and comparisons to be presented in the remaining of this section.

3.1 Illustration

The performance of the proposed method is shown in Fig. 3 for Pumadyn data with different values of the concavity parameter c. Two things happen when the value of c is increased. Firstly, the minimum validation errors deteriorate.

Table 1
Mean squared errors and standard deviations of the squared errors (in parentheses) on test data for networks, which have the minimum mean squared errors on validation data.

Method, $p(s)$	Pumadyn	Add10	Computer
Weight decay, s^2	0.52(0.74)	0.05(0.07)	0.02(0.05)
Input decay, $s^2/(\eta + s^2)$	0.39(0.56)	0.04(0.06)	0.02(0.05)
Proposed, $c \log(1 + s/c)$	0.38(0.54)	0.04(0.06)	0.02(0.05)

Table 2
The number of selected input variables (the 1st value) and hidden nodes (the 2nd value) in networks, which have the minimum mean squared errors on validation data.

Method, $p(s)$	Pumadyn	Add10	Computer
Weight decay, s^2	32 | 8	10 | 10	21 | 9
Input decay, $s^2/(\eta + s^2)$	2 | 3	8 | 8	21 | 10
Proposed, $c \log(1 + s/c)$	2 | 3	10 | 7	18 | 7

Secondly, the networks corresponding to these minimum validation errors become less penalized, that is, the minima are obtained with smaller values of the penalty parameter λ. As a consequence, more input variables become selected with large values of c. The difference in the number of selected input variables between the choices $c = 0.1$ and $c = 100$ is notable; three variables versus 21 variables. The networks with small values of c are able to use the selected input variables more efficiently.

Regardless of the value of c, the number of selected hidden nodes increases very rapidly when the value of λ goes below a certain level. However, for each value of c, the number of selected hidden nodes is three in all the networks, which have the smallest validation errors. The results of this illustrative experiment imply that the advantage of using a small value of c is twofold. Prediction accuracy increases and the number of selected input variables decreases.

3.2 Comparisons

The prediction errors on the test sets are presented in Table 1. All the methods are approximately equally accurate in the cases of Add10 and Computer activity data. On Pumadyn data, the proposed method and input decay have smaller errors than weight decay has, although the standard deviations are relatively large. The proposed method is also competitive in terms of prediction accuracy with several other nonlinear methods, which have been applied to Pumadyn data (Orr et al., 2000). However, the results are not fully comparable, since Orr et al. (2000) have a slightly different setup in the experiments.

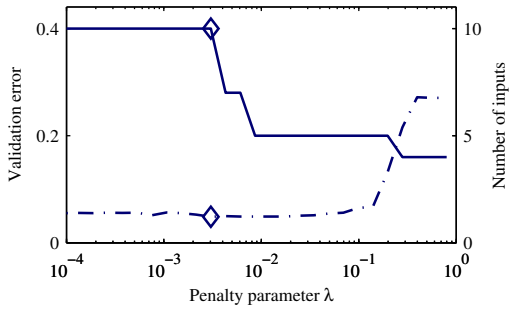

Fig. 4. Mean squared validation error (dash-dotted line) and the number of selected input variables (solid line) as a function of the penalty parameter λ for Add10 data. The concavity parameter has the value $c = 0.1$, but the curves look similar for the five other values as well. Markers indicate the position of the minimum mean squared validation error. The relevant five input variables are selected in the range of λ from 0.01 to 0.2.

Table 2 presents the number of selected input variables and the number of selected hidden nodes in the networks, which produce the minimum validation errors. The networks optimized with the weight decay penalization always include all the input variables. The outcoming weights of the hidden nodes are slightly below the threshold for two nodes and one node in the cases of Pumadyn and Computer data, respectively. Weight decay clearly produces more complex networks than the two other methods in terms of the number of effective units.

The proposed method and input decay suggest an identical architecture of the network for Pumadyn data. Input decay needs a few more hidden nodes than the proposed method does in the cases of Add10 and Computer data. Both methods have difficulties in discarding input variables from these two data sets. This is inconvenient, particularly because it is known that only the first five input variables are relevant in Add10 data. There exists a clear functional input-output mapping and the amount of noise is low, which makes the validation error curve flat below a certain value of λ, as shown in Fig. 4. Thereby, the problem concerns rather model selection than input selection. Choosing a larger λ for which the validation error is practically identical to the minimum value results in a network with only the relevant variables.

Tables 3 and 4 show further comparisons on Add10 data, where the number of samples and the degree of noise are changed. The parameters $c = 0.1$ and $\eta = 0.01$ are used for the proposed method and input decay, respectively. These data sets are harder to model than the original Add10 data and overfitting happens for small positive values of λ. For this reason, the minimum validation networks that have been penalized by the proposed and input decay functions are heavily pruned, but they mostly include the relevant five input variables. Input decay and the proposed method are more accurate than weight decay.

Table 3
Mean squared errors and standard deviations of the squared errors (in parentheses) on test data for networks, which have the minimum mean squared errors on validation data. The data set is Add10 with different numbers of samples N and different degrees of noise standard deviation σ.

Method, $p(s)$	$N = 100$ $\sigma = 1$	$N = 250$ $\sigma = 1$	$N = 500$ $\sigma = 1$	$N = 100$ $\sigma = 2$	$N = 250$ $\sigma = 2$	$N = 500$ $\sigma = 2$	$N = 100$ $\sigma = 3$	$N = 250$ $\sigma = 3$	$N = 500$ $\sigma = 3$
Weight decay, s^2	0.11(0.17)	0.07(0.12)	0.05(0.07)	0.31(0.47)	0.22(0.33)	0.22(0.35)	0.72(1.03)	0.37(0.55)	0.35(0.52)
Input decay, $s^2/(\eta + s^2)$	0.12(0.27)	0.05(0.07)	0.04(0.06)	0.29(0.40)	0.17(0.25)	0.18(0.26)	0.69(1.00)	0.43(0.66)	0.31(0.45)
Proposed, $c\log(1 + s/c)$	0.09(0.14)	0.06(0.09)	0.04(0.06)	0.25(0.39)	0.18(0.25)	0.18(0.26)	0.62(0.89)	0.33(0.49)	0.31(0.45)

Table 4
The number of selected input variables (the 1st value) and hidden nodes (the 2nd value) in networks, which have the minimum mean squared errors on validation data. The data set is Add10 with different numbers of samples N and different degrees of noise standard deviation σ.

Method, $p(s)$	$N = 100$ $\sigma = 1$	$N = 250$ $\sigma = 1$	$N = 500$ $\sigma = 1$	$N = 100$ $\sigma = 2$	$N = 250$ $\sigma = 2$	$N = 500$ $\sigma = 2$	$N = 100$ $\sigma = 3$	$N = 250$ $\sigma = 3$	$N = 500$ $\sigma = 3$
Weight decay, s^2	10 | 9	10 | 10	10 | 9	10 | 8	10 | 9	10 | 9	10 | 6	10 | 9	10 | 9
Input decay, $s^2/(\eta + s^2)$	5 | 4	6 | 5	5 | 6	8 | 3	7 | 4	9 | 5	4 | 9	3 | 10	7 | 4
Proposed, $c\log(1 + s/c)$	5 | 5	6 | 6	5 | 7	5 | 4	5 | 4	6 | 6	8 | 2	6 | 4	5 | 5

4 Conclusions

We have presented a method that penalizes the weights of a multilayer perceptron during the fitting process in such a way that unnecessary input variables and hidden nodes are removed from the network. Unlike the sequential pruning approaches, the proposed method is defined more directly as an optimization problem of continuous-valued parameters. The analyst starts by constructing a complex enough network that includes all the input variables. The amount of penalization and, optionally, the shape of the penalty function are the only free parameters that the analyst needs to tune based on validation data. Experimental results on three benchmark data sets show that the proposed penalization method results in an equivalent or better generalization capability than the weight decay and input decay penalization methods, yet being able to simplify the architecture of the network.

A Mathematical Description of the Multilayer Perceptron

Initialize $\boldsymbol{z}^{(0)} = \boldsymbol{x}$. The outcoming signals of successive layers can be defined by the recursion

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{\phi}^{(\ell)}([\boldsymbol{w}_1^{(\ell)}, \cdots, \boldsymbol{w}_{I(\ell)}^{(\ell)}]\boldsymbol{z}^{(\ell-1)} + \boldsymbol{w}_0^{(\ell)}) \tag{A.1}$$

for $\ell = 1, \ldots, L$. The complete network constitutes the model $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{z}^{(L)}$. The activation functions are differentiable and componentwise such that $[\boldsymbol{\phi}^{(\ell)}(\boldsymbol{u})]_i = \phi_i^{(\ell)}(u_i)$.

Acknowledgment

This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. Timo Similä is thankful for the personal support by the Nokia Foundation. This publication only reflects the authors' views.

References

Anders, U., Korn, O., 1999. Model selection in neural networks. Neural Networks. 12(2), 309–323.

Bakin, S., 1999. Adaptive regression and model selection in data mining problems. Ph.D. Dissertation, The Australian National University, Canberra, Australia.

Bi, J., Bennett, K.P., Embrechts, M., Breneman, C.M., Song, M., 2003. Dimensionality reduction via sparse support vector machines. Journal of Machine Learning Research. 3, 1229–1243.

Bishop, C.M., 1995. Neural Networks for Pattern Recognition, New York: Oxford University Press.

Blum, A.L, Langley, P., 1997. Selection of relevant features and examples in machine learning. Artificial Intelligence. 97(1–2), 245–271.

Boyd, S., Vandenberghe, L., 2004. Convex Optimization, Cambridge University Press.

Castellano, G., Fanelli, A.M., 2000. Variable selection using neural-network models. Neurocomputing. 31(1–4), 1–13.

Chapados, N., Bengio, Y., 2001. Input decay: Simple and effective soft variable selection. In: Proc. IEEE International Joint Conference on Neural Networks (IJCNN 2001). Vol 2, 1233–1237.

Fan, J., Li, R., 2001. Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American Statistical Association. 96(456), 1348–1360.

François, D., Rossi, F., Wertz, V., Verleysen, M., 2007. Resampling methods for parameter-free and robust feature selection with mutual information. Neurocomputing. 70(7–9), 1276–1288.

Friedman, J.H., 1991. Multivariate adaptive regression splines. The Annals of Statistics. 19(1), 1–67.

Gallinari, P., Cibas, T., 1999. Practical complexity control in multilayer perceptrons. Signal Processing. 74(1), 29–46.

Grandvalet, Y., Canu, S., 1999. Outcomes of the equivalence of adaptive ridge with least absolute shrinkage. Advances in Neural Information Processing Systems. Vol 11, Cambridge, MA: MIT Press, 445–451.

Guyon, I., Elisseeff, A., 2003. An introduction to variable and feature selection. Journal of Machine Learning Research. 3, 1157-1182.

Hassibi, B., Stork, D.G., 1993. Second order derivatives for network pruning: optimal brain surgeon. Advances in Neural Information Processing Systems. Vol 5, Cambridge, MA: MIT Press, 164–171.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks. 2(5), 359–366.

Hornik, K., Stinchcombe, M., White, H., 1990. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks. 3(5), 551–560.

Kohavi, R. John, G.H., 1997. Wrappers for feature subset selection. Artificial Intelligence. 97(1–2), 273–324.

Le Cun, Y., Denker, J.S., Solla, S.A., 1990. Optimal brain damage. Advances in Neural Information Processing Systems. Vol 2, Cambridge, MA: MIT Press, 598–605.

Li, K., Peng, J.-X., 2007. Neural input selection–A fast model-based approach. Neurocomputing. 70(4–6), 762–769.

Møller, M.F., 1993. A scaled conjugate gradient algorithm for fast supervised learning. Neural Networks. 6(4), 525–533.

Nguyen, D., Widrow, B., 1990. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Proc. IEEE International Joint Conference on Neural Networks (IJCNN 1990). Vol 3, 21–26.

Orr, M., Hallam, J., Murray, A., Leonard, T., 2000., Assessing RBF networks using DELVE. International Journal of Neural Systems. 10(5), 397–415.

Rivals, I., Personnaz, L., 2003. Neural-network construction and selection in nonlinear modeling. IEEE Transactions on Neural Networks. 14(4), 804–819.

Similä, T., 2007. Majorize-minimize algorithm for multiresponse sparse regression. In: Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2007). Vol 2, 553–556.

Similä, T., Tikka, J., 2007. Input selection and shrinkage in multiresponse linear regression. Computational Statistics & Data Analysis. 52(1), 406–422.

Steppe, J.M., Bauer, Jr., K.W., Rogers, S.K., 1996. Integrated feature and architecture selection. IEEE Transactions on Neural Networks. 7(4), 1007–1014.

Tikka, J., Lendasse, A., Hollmén, J., 2006. Analysis of fast input selection: Application in time series prediction. In: Proc. 16th International Conference on Artificial Neural Networks (ICANN 2006). Springer-Verlag. LNCS 4132, 161–170.

Tikka. J., 2007. Input selection for radial basis function networks by constrained optimization. In: Proc. 17th International Conference on Artificial Neural Networks (ICANN 2007). Springer-Verlag. LNCS 4668, 239–248.

White, H., 1990. Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. Neural Networks. 3(5), 535–549.

Yacoub, M., Bennani, Y., 2000. Features selection and architecture optimization in connectionist systems. International Journal of Neural Systems. 10(5), 379–395.

Yuan, M., Lin, Y., 2006. Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society, Ser. B. 68(1), 49–67.

Zhang, G.P, 2007. Avoiding pitfalls in neural network research. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews. 37(1), 3–16.