Partial Retraining:
A New Approach to Input Relevance Determination

Piërre van de Laar , Tom Heskes , and Stan Gielen
RWCP[*] Novel Functions SNN[†] Laboratory
Department of Medical Physics and Biophysics ,
University of Nijmegen ,
Nijmegen, The Netherlands
email: { pierre , tom , stan }@mbfys.kun.nl

Abstract

In this article we introduce partial retraining, an algorithm to determine the relevance of the input variables of a trained neural network. We place this algorithm in the context of other approaches to relevance determination. Numerical experiments on both artificial and real-world problems show that partial retraining outperforms its competitors, which include methods based on constant substitution, analysis of weight magnitudes, and "optimal brain surgeon".

1 Introduction

Feedforward neural networks are able to learn the relationship between input and output variables. Even when knowledge about the problem is limited, as for example in cases where no explicit physical or economical model can be built, neural networks may still capture some of the underlying principles. Especially with a lack of domain knowledge, the usual approach in neural network modeling is to include all input variables that may have an effect on the output. This approach is suboptimal in several aspects. First of all, the inclusion of irrelevant variables degrades generalization. Secondly, resources are wasted by measuring irrelevant variables. And finally, a model with irrelevant variables is more difficult to understand.

In this article we discuss how to determine the relevance of the input variables. Relevance information increases the user's understanding of the problem. Furthermore, removal of the irrelevant input variables reduces the complexity of the neural network, resulting in a better performing, more efficient, and better comprehensible neural network.

Before reviewing the relevance determination algorithms proposed in the literature (section 3), we will first, in section 2, give our definition of relevance. Partial retraining is introduced in section 4. In section 5, we will compare the various algorithms on a set of artificial and real-world problems. Our conclusions and some discussion can be found in section 6.

2 Relevance

2.1 How can one define relevance?

Many closely related definitions of relevance have been proposed (see e.g. [1, 2, 3]). Here we adopt the very general definition of the relevance $\mathcal{R}_i$ of variable i as the difference in performance on a

[*] Real World Computing Partnership
[†] Foundation for Neural Networks

task with and without input variable i, given all other input variables:

$$\mathcal{R}_i = \mathcal{P} - \mathcal{P}_{\{-i\}} \ , \tag{1}$$

with $\mathcal{P}$ and $\mathcal{P}_{\{-i\}}$ the optimal performance that can be achieved using all N input variables and using all input variables except input variable i, respectively. We propose to measure this performance by

$$\mathcal{P} = 1 - \frac{E}{E_{\text{total}}}$$

with E the smallest possible error given the available inputs and E_{total} the smallest possible error without any inputs. If the error function is the sum squared error, the total error is nothing but the variance in the output data. In this context the performance equals the coefficient of determination R^2 (or squared multiple correlation coefficient) [4] and, consequently, the relevance of variable i is the change in this coefficient.

Until now, we have defined relevance as a property belonging to a specific *task*. We are interested in classification and prediction tasks, where models need to be constructed for (approximate) solutions. Therefore, we would like to narrow down our definition of relevance to a definition which includes not only the task itself but also the class of models used to solve it. This we do by replacing in our definition of relevance the optimal performance by the optimal performance given the class of models. Furthermore, for mere notational convenience, we restrict ourselves to models with one output. Generalization to more outputs is straightforward.

2.2 How can one compute relevance?

The latter definition of relevance suggests the following approach to determine the relevance of all N input variables using neural networks.

- Train a neural network using all input variables and estimate its performance;
- For each input variable X_i: train a new neural network with all other $N-1$ input variables;
- Estimate the performance of these N networks and the corresponding relevances,

  $$\mathcal{R}_i = \mathcal{P}_w(X) - \mathcal{P}_{w^*_{\{-i\}}}(X_{\{-i\}}) \ ,$$

  where the subscripts w and $w^*_{\{-i\}}$ refer to the dependency of the performance on the (optimal) weights of the neural networks.

However, this approach has several serious drawbacks. Firstly, not all data can be used to train the network, because the estimation of the performance of a neural network should be based on the generalization performance on an independent test set. In this paper, we assume that the training error of a network that has not overfitted the data, yields a useful indication of the generalization error (see section 6 for more details and possible improvements). Secondly, training these N new neural networks is extremely time consuming. Thirdly, neural networks are notoriously unstable: for example starting from slightly different initial conditions networks may end up at completely different solutions. This instability adds a significant noise factor to the various performances, making it much more difficult to compare them. And finally, this approach seems to be rather inefficient: from all information implicitly available in the weights of the network trained using all input variables, the procedure only takes into account the network's performance. Faster and more reliable algorithms are therefore not only desirable but (probably) also obtainable. In fact, the relevance determination algorithms, which will be described in the next section, do not train N new neural networks, but use stable and fast procedures to arrive at these N networks starting from the original network.

3 Relevance determination algorithms

In this section, we will give an overview of the algorithms proposed in the literature that can be used for relevance determination. Some of these algorithms were initially introduced for sensitivity analysis. Sensitivity analysis measures changes in the performance of a single model as a function of changes in input variables. According to our definition, relevance determination judges the difference between two models: a model trained with and a model trained without a particular input variable. In its training phase the latter model can try to compensate for the lack of this input variable. Sensitivity analysis is identical to relevance determination under the assumption that no such compensation takes place.

In our survey of the literature we will focus on the similarities and main differences between the various ideas that are used to determine relevance. The small difference caused by using the median, modus, or midpoint of the range instead of the average value, or the range, or quartiles instead of the standard deviation are neglected. Furthermore, we try to describe and implement all algorithms in terms of changes in performance, although some of them were introduced to measure output changes. Again, this generalization is justified since, for a model that has not been overfitted, output changes are highly correlated to performance changes.

We will divide the relevance determination algorithms into four groups: data modification, missing values, approximate retraining, and other approaches.

3.1 Data modification

All algorithms described in this subsection modify the data, i.e., they perform some kind of sensitivity analysis. They are based on the idea that an input variable has to be irrelevant if changing its value does not affect the model's performance. The performance without an input variable X_i is estimated by the performance with a modified value X_{mod_i} of this input variable:

$$\mathcal{P}_{w^*_{\{-i\}}}(X_{\{-i\}}) \approx \mathcal{P}_w(X_{\{-i\}}, X_{\mathrm{mod}_i}) \ ,$$

where the subscripts w and $w^*_{\{-i\}}$ refer to the dependency of the performance on the (optimal) weights of the neural networks. The time needed by these algorithms to determine the relevance of a single variable is thus (almost) equal to the time needed to process a dataset by the neural network.

The data modification algorithms can be separated into three different groups. "Constant substitution" substitutes a constant value for the input variable under investigation [2, 3, 5, 6, 7, 8], "translation factor" modifies the data by translation [7, 9, 10], and "data permutation" permutes the data of input variable i across patterns [11].

3.2 Missing values

The following algorithms treat the removed input variable as a missing value and approximate the performance without an input by the performance of the network with a substitution for the missing input variable i based on all other inputs [$X_{\mathrm{mis}_i} = h(X_{\{-i\}})$], i.e.,

$$\mathcal{P}_{w^*_{\{-i\}}}(X_{\{-i\}}) \approx \mathcal{P}_w(X_{\{-i\}}, X_{\mathrm{mis}_i}) \ .$$

The time needed by these algorithms to determine the relevance of a single variable is thus equal to the time needed to estimate the missing value and the time needed to process the data by the neural network. The algorithms based on this idea can be subdivided into three groups based on the dependencies assumed to exist between the input variables: independent, linearly dependent, and nonlinearly dependent.

Under the assumption that all input variables are independent, the remaining inputs yield no information about the missing value. A usual procedure then is to replace the missing value by the average value of this variable [2, 5]. This algorithm, which we will call "average substitution", is thus equivalent to constant substitution where the substituted constant is the average value (see subsection 3.1).

The assumption that the missing value depends linearly on the other inputs yields an algorithm called "linear substitution" see, for example, [12] and [13]. The linear transformation needed for the reconstruction of this missing value can be extracted from the training set by solving

$$\mathcal{T} = \underset{B}{\text{argmin}} \sum_{\mu} \|X^{\mu} - B{X_{\{-i\}}}^{\mu}\|^2 , \tag{2}$$

with X the complete input, $X_{\{-i\}}$ the incomplete input without input variable i, μ labels the examples, and where B and $\mathcal{T}$ are linear transformations.

In general, it can also be assumed that the missing value depends in a nonlinear way on the other inputs. In [14], for example, it is proposed to train a neural network to find this nonlinear relationship. This approach, however, is hardly an improvement over the straightforward approach for relevance determination described in section 2: it also requires training N neural networks, which makes it time-consuming, instable, and inefficient. Other algorithms to estimate missing values, for example, Parzen windows, see e.g. [15], and k-nearest neighbor, see e.g. [16], can be applied in a similar manner, but have not been included in our simulations.

3.3 Approximate retraining

The algorithms described in this subsection approximate the weights ${w^*}_{\{-i\}}$ one would get when one would train a new network using only $N-1$ input variables. The weights $\tilde{w}_{\{-i\}}$, which are the result of this approximation, are in general a function of the old weights and the training patterns. The performance corresponding to ${w^*}_{\{-i\}}$, i.e., training a new network with one input less, is estimated using the weights $\tilde{w}_{\{-i\}}$ by

$$\mathcal{P}_{{w^*}_{\{-i\}}}(X_{\{-i\}}) \approx \mathcal{P}_{\tilde{w}_{\{-i\}}}(X_{\{-i\}}) .$$

Approximate retraining algorithms have a flavor of complexity reduction algorithms (see e.g. [17, 18]). Each reduction step, which removes the least relevant weight or set of weights, requires an estimate of the change in performance due to this reduction. Approximate retraining can be viewed as a one-step complexity reduction algorithm where the set of weights to be removed consists of all outgoing weights of a particular input unit.

As an example, we consider optimal brain surgeon (OBS) [19] for multi-layered perceptrons which removes the least relevant weight based on a second order approximation of the error function. Just like optimal cell damage (OCD) [20] is a generalization of optimal brain damage (OBD) [21] which computes the effect of removing a whole input unit, optimal brain surgeon can also easily be generalized (see for example [22]). In that case the approximate weights $\tilde{w}_{\{-i\}}$, given that input variable i is removed, obey

$$\tilde{w}_{\{-i\}} = w - \mathcal{H}^{-1} e_i \left(e_i^T \mathcal{H}^{-1} e_i\right)^{-1} {w^1}_i ,$$

yielding an increase in error given by

$$\tilde{E} - E = \frac{1}{2} {w^1}_i^T \left(e_i^T \mathcal{H}^{-1} e_i\right)^{-1} {w^1}_i ,$$

with Hessian matrix $\mathcal{H} \equiv \frac{\partial^2 E}{\partial w^2}$, matrix $e_i \equiv (e_{1i}, e_{2i}, \ldots, e_{Mi})$ where e_{ji} stands for the unit vector in weight space corresponding to weight ${w^1}_{ji}$ between hidden unit j of the first hidden layer and input variable i, column vector ${w^1}_i \equiv \left({w^1}_{1i}, {w^1}_{2i}, \ldots, {w^1}_{Mi}\right)^T$, and the superscript T to denote transpose.

Some of the previously described algorithms also correspond to approximate retraining algorithms. Constant substitution corresponds to an approximate retraining algorithm which changes only the thresholds of the units of the first hidden layer, since the effect of a constant input is mathematically equivalent to a shift in the values of the thresholds of the hidden units. Linear substitution can be mapped onto an approximate retraining algorithm which only changes the weights between input and the first hidden layer. Partial retraining, the algorithm which we propose in section 4, can be seen as an extension hereof and also belongs to the category of approximate retraining algorithms.

3.4 Other approaches

In this subsection we describe three classes of algorithms that do not fit into our general framework. These algorithms propose to compute quantities not directly related to relevance. The assumption underlying these algorithms is that the ordering of the input variables based on these quantities is close to the ordering of the input variables based on relevance.

3.4.1 Derivative information

The algorithms in this category are similar to the data modification algorithms described in subsection 3.1. Instead of perturbing the inputs, they try to estimate the effect of these variations by computing derivatives. Unfortunately, the most obvious choice, computing the derivative of the performance itself, does not make sense since at a minimum of the error function with respect to the weights

$$\frac{\partial \mathcal{P}(X)}{\partial X_i} = 0 \ ,$$

for *any* input variable, i.e., not just for irrelevant ones. An alternative, as suggested in for example [23, 24, 25], is to extract “sensitivity” or “saliency” information from the output derivatives for single patterns. Since the calculation of the derivative for a single pattern can be done similarly to backpropagation, these algorithms will take about the same time as a single step of batch learning of a neural network to determine the relevance of a single variable. In our simulations, see section 5, we will consider the sum of the absolute value of the derivatives of the output of single patterns to the input variables [23, 25, 26] and we will refer to this quantity as “absolute derivative”.

3.4.2 Weight analysis

Whereas most of the other algorithms treat a neural network as a black box and can in principle be applied to any classification or prediction model, the algorithm in [1, 27] really looks into the network and gives an interpretation of the weights. For multi-layered perceptrons, it defines the importance of the information flowing from unit i to unit j as

$$\mathcal{I}_{ji} = \frac{|w_{ji}|}{\sum_{i'} |w_{ji'}|} \ ,$$

where the sum is over all incoming weights $(w_{ji'})$ of neuron j. The importance of an input variable i for the output can be found by propagating these importances through the network. For a two-layered perceptron we obtain

$$\mathcal{I}_i = \sum_{j}^{M} \frac{|w^1{}_{ji}|}{\sum_{i'}^{N} |w^1{}_{ji'}|} \frac{|w^2{}_j|}{\sum_{j'}^{M} |w^2{}_{j'}|} \ ,$$

where $w^1{}_{ji}$ denotes the weight between input variable i and hidden unit j, and $w^2{}_j$ between hidden unit j and the output. Note that weight analysis depends only indirectly on the data through the value of the weights, unlike all other relevance determination algorithms, which make explicit use of the data. So the computational load of weight analysis is very small compared to the other algorithms, especially for large datasets.

3.4.3 Automatic relevance determination

The automatic relevance determination (ARD) model [28] is a Bayesian model, whose prior over the regression parameters embodies the concept of relevance. A regularization constant is introduced for each input variable or, in other words, each input variable is given its own weight decay parameter. ARD then searches for the regularization constants that maximize the so-called evidence [28]. According to ARD, the largest inferred regularization constant corresponds to the least relevant input variable.

4 Partial retraining

In this section, we propose a new algorithm, which we call "partial retraining". Partial retraining can be derived by assuming that a neural network trained on all N input variables has constructed a good representation of the data in its hidden layers. The goal is to find a new neural network, based on $N-1$ input variables, with hidden-layer activities as close as possible to the original ones.

In the first step, partial retraining determines the new weights between the $N-1$ input variables ($X_{\{-i\}}$) and the first hidden layer using the original incoming activity of the first hidden layer $h_1 \equiv w^1 X$ through

$$\tilde{w}^1_{\{-i\}} = \underset{B}{\text{argmin}} \sum_\mu \|h_1^\mu - B X^\mu_{\{-i\}}\|^2 \ , \tag{3}$$

where μ labels the examples. The difference between fitting the incoming activity and fitting the outgoing activity of the hidden layer is almost negligible (see e.g. [29]). We prefer fitting the incoming activity, since this least squares problem can be easily solved by matrix inversion or conjugate gradient (see for example [13]). Furthermore, it can be easily shown [see equations (2) and (3)] that

$$\tilde{w}^1_{\{-i\}} = w^1 \mathcal{T} \ ,$$

i.e., the new weights between the input and the first hidden layer are chosen such that the neural network estimates the missing value based on linear dependencies, and processes the "completed" input data.

The compensation of the errors introduced by removing an input variable is probably not perfect due to noise and nonlinear dependencies in the data. Therefore, to further minimize the effects caused by the removal of an input variable, the new weights between hidden layers λ and $\lambda - 1$ are calculated from

$$\tilde{w}^\lambda_{\{-i\}} = \underset{B}{\text{argmin}} \sum_\mu \|h_\lambda^\mu - B \tilde{H}^\mu_{\lambda-1}\|^2 \ ,$$

where $h_\lambda = w_\lambda H_{\lambda-1}$ is the original incoming activity of hidden layer λ and $\tilde{H}_\lambda$ is the new λ^{th} hidden layer activity without input variable i and the new estimated weights.

Finally, the weights between the output and the last hidden layer are re-estimated. Although we could treat the output layer similarly to the hidden layers, we suggest a different approach. Since the desired output is given by the data, we can directly calculate the desired incoming activity of the output layer by applying the inverse of the output's transfer function:

$$\tilde{w}^{K+1}_{\{-i\}} = \underset{B}{\text{argmin}} \sum_\mu \|f_O^{-1}(T^\mu) - B \tilde{H}^\mu_K\|^2 \ ,$$

where f_O^{-1} is the inverse of the transfer function of the output layer, $\tilde{H}_K$ is the activity of last hidden layer K given the new weights and the new, incomplete input, and T^μ is the desired output.

Summarizing, partial retraining simplifies the hard problem of training a neural network by introducing additional variables (the activities of the hidden units), similarly to the Expectation Maximization algorithm [30]. But, unlike the EM algorithm, the values of these variables are calculated from the original network, instead of inferred from the current network. Given these additional variables, partial retraining determines the weights of the neural network by solving a least squares problem for each layer.

Partial retraining can be seen as a combination of a relevance determination algorithm which estimates a missing value and an algorithm which estimates retraining. Its calculation time is (roughly) equal to the number of layers multiplied by the time needed to estimate the new weights[a] and the time needed to process a dataset by a single layer neural network.

[a] The time needed to estimate the new weights is (almost) equal to the time needed to calculate a missing value.

5 Simulations

5.1 General description

The quality of a particular algorithm for relevance determination can only be established by its performance in practice. We define the quality $\mathcal{Q}$ of an algorithm for a particular number of remaining input variables $N-n$ as the performance given the suggested subset of input variables divided by the performance corresponding to the optimal subset of input variables:

$$\mathcal{Q} = \frac{\mathcal{P}_{N-n}}{\mathcal{P}_{\mathrm{opt}_{N-n}}} \ . \tag{4}$$

For artificial datasets, the optimal performance can be calculated exactly. But, in real-world problems, we do not know which variables are relevant and which are not. To get as close as possible to our definition (4), we propose to divide each real-world dataset in a training and test set and to estimate the optimal performance, by training, for each combination of input variables, hundred networks on the training set and averaging over the performance of these networks on the test set.

For all simulations in this article, we used a two-layered feedforward neural network, with the hyperbolic tangent and the identity as transfer functions of the hidden and output layer respectively. Starting from small random initial values, weights were updated using backpropagation on the sum of squared errors. Training was stopped at the minimum of the error on a set of validation patterns (except in rule-plus-exception where we had only 64 training patterns available and we stopped based on the convergence of the error on the training set). We made sure that on each dataset we had hundred well-trained and good generalizing networks. For a fair comparison, the relevance determination algorithms were applied on the same hundred networks and we estimated the relevance of input variables using both the training and validation patterns. Automatic relevance determination [28] is part of an integrated Bayesian framework. It cannot be applied on trained networks in a manner similar to the other algorithms. Therefore, we have not included ARD in these simulations.

As mentioned in subsection 3.4, not all algorithms try to estimate the relevance itself, but all of them claim to be able to distinguish relevant from irrelevant variables. Therefore, for each of the hundred networks, each algorithm had to determine the least relevant variable. We removed this variable, adjusted the network as described below, and asked for the next variable to be removed. This iterative procedure is necessary since the relevance of a variable may change due to the removal of another variable. Consider for example the extreme situation in which two variables contain the same relevant information. Both are individually irrelevant since no information is lost by removing either one of them. However, after one of them has been removed, the relevance of the other one increases dramatically. The iterative procedure which starts with all variables and which eliminates one variable at a time, is called "backward elimination" [31]. Unfortunately, backward elimination does not guarantee that subsequently removing the n least relevant variables necessarily yields the optimal subset with $N-n$ variables. The chosen selection strategy does not affect our comparison for two reasons. First, because the optimal subsets for the specific artificial problems in this paper *can* be found by subsequently removing the least relevant variables. Second, if this were not the case, as in our real-world problem, all relevance determination algorithms are in principle equally hampered.

After each removal of an input variable, we have to adjust the network to be in accordance with the remaining variables. For most algorithms, we can simply take the smaller network which has been constructed in the first place to compute the relevance of the left-out input variable. Only for data modification using the "translation factor" and "data permutation", and for the "absolute derivative", it is not straightforward how to continue with one variable less. We decided to treat them similarly to the other sensitivity-related algorithms and substituted the average value of the removed variable.

Table 1: Data sets on which the various relevance determination algorithms were tested.

dataset	task	type of input variables	number of input variables	number of training patterns	number of validation patterns
Rule-plus-exception	classification	binary	6	64	0
Friedman	regression	continuous	10	200	200
Boston housing	regression	continuous	6	304	76

Table 2: Performance of the algorithms on the rule-plus-exception problem. The number of networks (out of a hundred) in which the two irrelevant input variables were indeed the first to be removed (middle column) and in which the two most relevant input variables were indeed the last to be removed (right column).

method	first two correct	last two correct
constant substitution	99	99
translation factor	99	100
data permutation	100	100
average substitution	99	100
linear substitution	99	100
optimal brain surgeon	98	82
absolute derivative	0	48
weight analysis	99	100
partial retraining	99	100

5.2 Data sets

The algorithms were tested on rule-plus-exception [3], Friedman [32], and Boston housing [33]. See Table 1 and the following paragraphs for additional information about these datasets.

5.2.1 Rule-plus-exception

Our first artificial dataset is a six-dimensional version of the rule-plus-exception problem [3]. The relevances of the six binary input variables, $X_1, \ldots, X_6$, of this classification problem differ significantly. As in the original problem, the binary output is given by

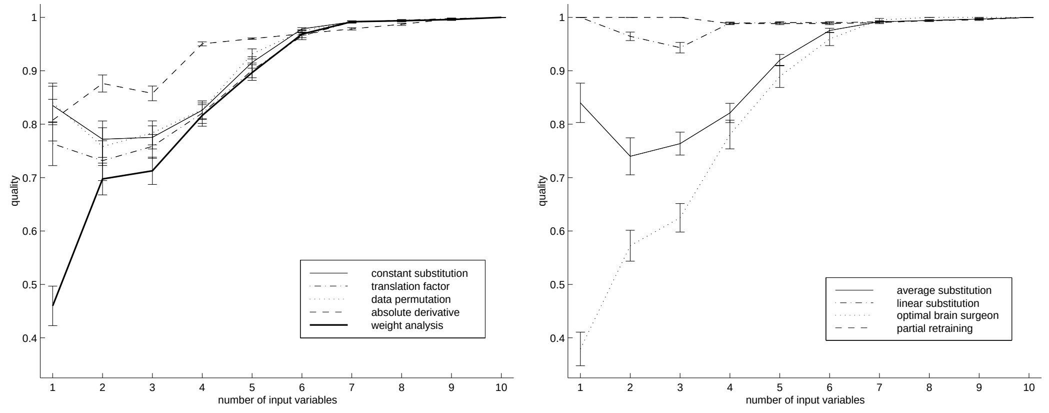
$$T = X_1 X_2 + \overline{X_1}\,\overline{X_2}\,\overline{X_3}\,\overline{X_4}\ ,$$

or in words, the output T is true if X_1 and X_2 are both true and, in the special case, when X_1, X_2, X_3, and X_4 are all false. Note that the output is independent of X_5 and X_6.

We trained neural networks with six inputs, one output, and two hidden units, similar to the architecture of [3], on all $2^6 = 64$ possible training patterns. In table 2, we give the frequencies that the irrelevant input variables (X_5 and X_6) were the first and the most relevant inputs (X_1 and X_2) the last to be removed.

5.2.2 Friedman

The second dataset is based on an example in [32]. This dataset has ten input variables, $X_1, \ldots, X_{10}$ which are uniformly distributed over $[0, 1]$. The response is given according to the following signal



(a) (b)

Figure 1: The qualities of relevance determination algorithms on the [32] dataset, in which eight of the ten input variables are independent and only four input variables are necessary to obtain the optimal performance.

plus noise model

$$T = 10\sin(\pi X_1 X_2) + 20\left(X_3 - \frac{1}{2}\right)^2 + 10X_4 + 5X_5 + \epsilon \ ,$$

where ϵ is $\mathcal{N}(0,1)$, i.e., standard normally distributed noise. The response does not depend on X_6, X_7, X_8, X_9, and X_{10}.

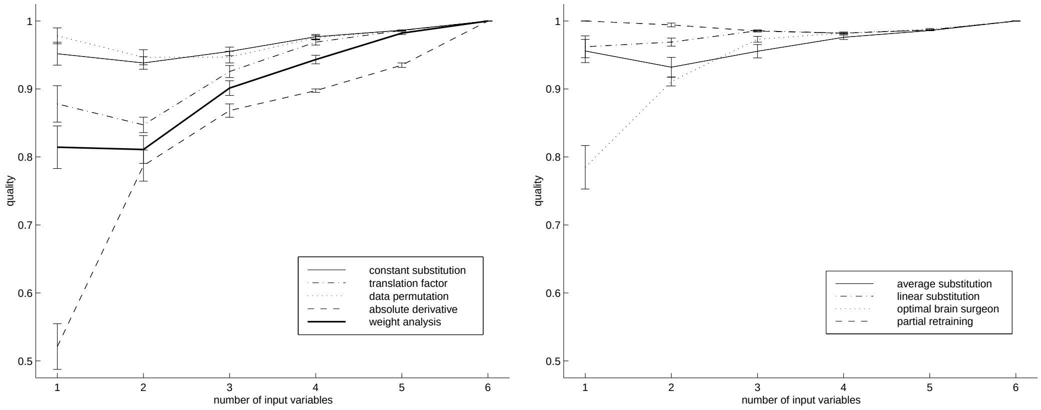
To make the dataset even more interesting for relevance determination, we assume, unlike [32], that only eight of the ten input variables are independent. Two irrelevant inputs are chosen identical, $X_9 \equiv X_{10}$, as well as two relevant inputs, $X_4 \equiv X_5$. As explained in subsection 5.1, X_4 is irrelevant given all other input variables and so is X_5. Therefore, only four variables are needed to obtain the optimal performance.

We used neural networks with ten inputs, five hidden units, and one output. For this artificial dataset, the performance for any subset of input variables can be computed exactly. Fig. 1 shows for each algorithm and for each number of remaining input variables the average and standard deviation of the qualities as defined in (4).

5.2.3 Boston housing

To test the different algorithms on a real-world problem we have selected the Boston housing dataset [33]. Although the Boston housing dataset has thirteen input variables, we only used six variables to prevent the explosion of possible subsets (2^6 instead of 2^{13} possible subsets). We kept the per capita crime rate by town (CRIM), nitric oxides concentration squared (NOXSQ), average number of rooms per dwelling (RM), index of accessibility to radial highways (RAD), full-value property-tax rate (TAX), and the percent of lower status of the population (LSTAT) to predict the median value of owner-occupied homes (MV).

The neural networks consisted of six inputs, four hidden units, and one output. The dataset was divided into a set of 380 patterns used for training and validation and a set of 126 patterns used for testing. For each subset of input variables, we trained hundred networks and computed their performance on the test set. We estimated the optimal performance given this particular subset through the average test performance of these hundred networks. Using the remaining 380 patterns we applied the usual procedure to determine the relevance of the input variables. Based



(a)

(b)

Figure 2: The qualities of relevance determination algorithms on the Boston housing dataset for one to six input variables left.

on the (estimated) optimal performances and the orderings of the input variables suggested by the various algorithms, we calculated the corresponding qualities and depicted these in Fig. 2.

5.3 Results

From our simulations, we can conclude the following.

- Especially in the rule-plus-exception problem (see Table 2), but also on the Boston housing dataset (see Fig. 2), absolute derivative comes out worst. This performance could have been expected because, as already mentioned in paragraph 3.4.1, it is not clear how to draw a quantitative link between information about derivatives and relevance. Also for measures based on derivative information other than absolute derivative, we did not arrive at significantly better results.

- Sensitivity-based measures, such as constant substitution, translation factor, data permutation and weight analysis, are fooled by correlations between input variables. For example, the inferior quality in Fig. 1 is due to the fact that these algorithms tend to keep both variables X_4 and X_5, which are both sensitive but redundant, and thus each irrelevant given the other.

- Optimal brain surgeon breaks down after several iterations (see Fig. 1). The theory behind optimal brain surgeon requires the networks to be close to a minimum of the error function on the training set, which was not the case in our experiment since networks were trained using cross-validation, resulting in networks close to a minimum of the error function on the validation set. Furthermore, it is well-known that after removal of several weights the approximations made by optimal brain surgeon become invalid and full retraining is necessary [19].

- Partial retraining and linear substitution clearly outperform all other algorithms. Partial retraining is better than linear substitution. Apparently, a faithful reconstruction of the data representation in the hidden layer of the original network yields a close approximation of a network that could be obtained in case of full retraining with one variable less.

6 Discussion

In this article we proposed partial retraining and contrasted it with other relevance determination algorithms. Based on the performance of these algorithms on artificial and real-world problems, we concluded that partial retraining outperforms all other relevance determination algorithms studied in this paper.

If a neural network is applied to predict or classify new examples, it should generalize well. To reflect this task, the relevance should be based on an independent test set and not, as is done in our simulations, on the training set. However, the use of an independent test set is often not desired especially not when data is hard or expensive to acquire, in which case the data should be used more effectively than for mere validation. Fortunately, our simulations show that when overfitting is avoided, relevance determination can be based on a training set and does not have to waste valuable data for a test set. However, a test set can be very useful to determine when to stop removing variables. Several suggestions in this direction can be found in the literature, both on pruning algorithms (see e.g. [17]) and on subset selection (see e.g. [4, 31]).

The relevance of information is also influenced by the effort needed to extract this information [34]. Effort is a negative factor: other things being equal, the greater the effort, the lower the relevance. In this article, we have assumed that the effort needed to extract the information of each input is equal. We can incorporate the effort needed to extract the information by modifying equation (1) to

$$\mathcal{R}_i = k_i(\mathcal{P} - \mathcal{P}_{\{-i\}})$$

where the same notation is used as in equation (1) and with k_i the effort associated with input variable i.

Another straightforward generalization of partial retraining is to consider not only inputs, but also hidden units. These hidden units can be viewed as input units of a smaller network [23, 35]. By detecting and removing the least relevant unit in the whole network, partial retraining is a fast and reliable method for architecture selection.

Partial retraining has been derived from the assumption that the hidden units of a network trained on all input variables provide a suitable data representation for solving the task. For multi-layered perceptrons, the type of neural networks considered in this paper, partial retraining is almost equivalent to linear substitution: treating the left-out input variable as a missing value which is approximated by a linear combination of the remaining input variables. Of course, there are other ways for computing missing value estimates, for example, Parzen windows, see e.g. [15] and k-nearest neighbor, see e.g. [16]. The close correspondence between partial retraining and linear substitution, however, implies that for relevance determination using multi-layered perceptrons linear substitution is the most obvious choice among algorithms based on missing value estimates. Partial retraining can also be applied to other types of networks, such as radial-basis function networks, but for these types of architectures an interpretation in terms of missing values is no longer possible.

Acknowledgements

We would like to thank Bert Kappen, Monique van de Laar-Nas, Marcel Nijman, Machiel Westerdijk, and Wim Wiegerinck for their useful comments on an earlier version of this paper.

References

[1] G. David Garson. Interpreting neural-network connection weights. *AI Expert*, 6(4):47–51, 1991.

[2] John Moody and Joachim Utans. Principled architecture selection for neural networks: application to corporate bond rating prediction. In John E. Moody, Steven J. Hanson, and

Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4: Proceedings of the 1991 Conference*, pages 683–690, San Mateo, 1992. Morgan Kaufmann.

[3] Michael C. Mozer and Paul Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

[4] Norman Richard Draper and Harry Smith. *Applied Regression Analysis.* Wiley Series in Probability and Mathematical Statistics. Wiley, New York, second edition, 1981.

[5] Chalapathy Neti, Eric D. Young, and Michael H. Schneider. Neural network models of sound localization based on directional filtering by the pinna. *Journal of the Acoustical Society of America*, 92(6):3140–3156, 1992.

[6] C. Aldrich and J. S. J. van Deventer. Modelling of induced aeration in turbine aerators by use of radial basis function neural networks. *The Canadian Journal of Chemical Engineering*, 73(6):808–816, 1995.

[7] R. Lacroix, K. M. Wade, R. Kok, and J. F. Hayes. Prediction of cow performance with a connectionist model. *Transactions of the American Society of Agricultural Engineers*, 38(5):1573–1579, 1995.

[8] Rudy Setiono and Huan Liu. Neural-network feature selector. *IEEE Transactions on Neural Networks*, 8(3):654–662, May 1997.

[9] J. Nijskens, D. de Halleux, and J. Deltour. Sensitivity study of a greenhouse climate dynamic model. *Bulletin des Recherches Agronomiques de Gembloux*, 26(3):389–410, 1991.

[10] R. L. Korthals, G. L. Hahn, and J. A. Nienaber. Evaluation of neural networks as a tool for management of swine environments. *Transactions of the American Society of Agricultural Engineers*, 37(4):1295–1299, 1994.

[11] William G. Baxt. Analysis of the clinical variables driving decision in an artificial neural network trained to identify the presence of myocardial infarction. *Annals of Emergency Medicine*, 21(12):1439–1444, 1992.

[12] Michael Egmont-Petersen. *Specification and assessment of methods supporting the development of neural networks in medicine.* PhD thesis, Maastricht University, Maastricht, 1996.

[13] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, 8(3):519–531, May 1997.

[14] A. Gupta and M. S. Lam. Estimating missing values using neural networks. *Journal of the Operational Research Society*, 47(2):229–238, 1996.

[15] Volker Tresp, Ralph Neuneier, and Subutai Ahmad. Efficient methods for dealing with missing data in supervised learning. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems 7: Proceedings of the 1994 Conference*, Cambridge, 1995. MIT Press.

[16] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[17] Russell Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[18] Tautvydas Cibas, Françoise Fogelman Soulié, Patrick Gallinari, and Sarunas Raudys. Variable selection with neural networks. *Neurocomputing*, 12(2-3):223–248, 1996.

[19] Babak Hassibi, David G. Stork, Gregory Wolff, and Takahiro Watanabe. Optimal Brain Surgeon: Extensions and performance comparisons. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems 6: Proceedings of the 1993 Conference*, pages 263–270, San Francisco, 1994. Morgan Kaufmann.

[20] Tautvydas Cibas, Françoise Fogelman Soulié, Patrick Gallinari, and Sarunas Raudys. Variable selection with optimal cell damage. In Maria Marinaro and Pietro G. Morasso, editors, *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 727–730. Springer-Verlag, 1994.

[21] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal Brain Damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2: Proceedings of the 1989 Conference*, pages 598–605, San Mateo, 1990. Morgan Kaufmann.

[22] Achim Stahlberger and Martin Riedmiller. Fast network pruning and feature extraction using the unit-OBS algorithm. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference*, pages 655–661, Cambridge, 1997. MIT Press.

[23] Kevin L. Priddy, Steven K. Rogers, Dennis W. Ruck, Gregory L. Tarr, and Matthew Kabrisky. Bayesian selection of important features for feedforward neural networks. *Neurocomputing*, 5(2/3):91–103, 1993.

[24] Gilbert L. Molas and Fumio Yamazaki. Neural networks for quick earthquake damage estimation. *Earthquake Engineering and Structural Dynamics*, 24(4):505–516, 1995.

[25] R. Naimimohasses, D. M. Barnett, D. A. Green, and P. R. Smith. Sensor optimization using neural network sensitivity measures. *Measurement Science & Technology*, 6(9):1291–1300, 1995.

[26] Jean M. Steppe and Kenneth W. Bauer, Jr. Improved feature screening in feedforward neural networks. *Neurocomputing*, 13(1):47–58, September 1996.

[27] Louis W. Glorfeld. A methodology for simplification and interpretation of backpropagation-based neural network models. *Expert Systems With Applications*, 10(1):37–54, 1996.

[28] David J. C. MacKay. Bayesian non-linear modelling for the predicition competition. In *ASHRAE Transactions*, volume 100, pages 1053–1062, Altlanta Georgia, 1994. ASHRAE.

[29] John O. Moody and Panos J. Antsaklis. The dependence identification neural network construction algorithm. *IEEE Transactions on Neural Networks*, 7(1):3–15, 1996.

[30] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[31] R. R. Hocking. The analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.

[32] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–141, 1991.

[33] David A. Belsley, Edwin Kuh, and Roy E. Welsch. *Regression diagnostics: identifying influential data and sources of collinearity.* Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1980.

[34] Dan Sperber and Deirdre Wilson. *Relevance: Communication and Cognition.* Basil Blackwell Ltd, Oxford, 1986.

[35] Thomas Czernichow. Architecture selection through statistical sensitivity analysis. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks - ICANN 96*, volume 1112 of *Lecture Notes in Computer Science*, pages 179–184. Springer, 1996.