Input Selection with Partial Retraining

Pièrre van de Laar, Stan Gielen, and Tom Heskes

RWCP* Novel Functions SNN** Laboratory,
Dept. of Medical Physics and Biophysics, University of Nijmegen, The Netherlands.

Abstract. In this article, we describe how input selection can be performed with partial retraining. By detecting and removing irrelevant input variables, resources are saved, generalization tends to improve, and the resulting architecture is easier to interpret. In our simulations, the relevant input variables were correctly separated from the irrelevant variables for a regression and a classification problem.

1 Introduction

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. Furthermore, the number of parameters of the neural network is chosen large, since with too few parameters a network is unable to learn a complex relationship between input and output variables. This approach is suboptimal because the inclusion of irrelevant variables and the (possibly) abundant parameters tends to degrade generalization. Secondly, resources are wasted by measuring irrelevant variables. And finally, a model with irrelevant variables and parameters is more difficult to understand.

Architecture selection algorithms (see e.g. [1,5,6,9]) try to remove irrelevant parameters and/or input variables and, consequently, save resources, improve generalization, and yield architectures which are easier to interpret. In this article, we will describe a new algorithm to perform input selection, a subproblem of architecture selection, which exploits partial retraining [11].

We will first, in section 2, describe partial retraining. Second, in section 3, we will describe how the performance of networks with different sets of input variables are compared. Then, we will briefly describe the problem of input selection and we will introduce our algorithm to perform input selection which exploits partial retraining in section 4. In section 5, we will perform simulations in which our algorithm is applied for input selection in two artificial problems. Our conclusions and some discussion can be found in section 6.

2 Partial retraining

Suppose we have a neural network which has been trained on $N$ input variables and we would like to determine the performance which can be achieved with

* Real World Computing Partnership
** Foundation for Neural Networks
a subset of these $N$ input variables. A naïve method is to train a new neural network on this subset to determine this performance. Instead of the computationally expensive process of training a new network, this new network might also be estimated based on the original network. Partial retraining [11] assumes that the neural network trained on all $N$ input variables has constructed a good representation of the data in its hidden layers needed to map the input to the output. Therefore, the new network is estimated by partial retraining such that its hidden-layer activities are as close as possible to the hidden-layer activities of the original neural network, although its input is restricted. The performance of this constructed network is an estimate of the performance we want to determine.

Partial retraining uses the following three steps to construct the network which only receives a subset of all variables as its input. In the first step, partial retraining determines the new weights between the subset of input variables and the first hidden layer by

$$
\tilde{w}_1 = \arg\min_B \sum_{\mu} \| h_{1\mu} - BX^{*\mu} \|^2,
$$

where $h_{1\mu} \equiv w_1 X$ denotes the original incoming activity of the first hidden layer, $X^{*\mu}$ stands for the input limited to the subset, and $\mu$ labels the patterns.

The difference between fitting the incoming activity and fitting the outgoing activity of the hidden layer is almost negligible (see e.g. [8]). We prefer fitting the incoming activity, since this can be done by simple and fast quadratic optimization. Furthermore, it can be easily shown [11] that the new weights between the input and the first hidden layer are chosen such that the neural network estimates the missing value(s) based on linear dependencies, and processes the “completed” input data.

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

$$
\tilde{w}_\lambda = \arg\min_B \sum_{\mu} \| h_{\lambda\mu} - B \tilde{H}_{\lambda-1\mu} \|^2,
$$

with $h_{\lambda\mu} \equiv w_\lambda H_{\lambda-1\mu}$ the original incoming activity of hidden layer $\lambda$ and $\tilde{H}_{\lambda\mu}$ the new $\lambda^{th}$ hidden layer activity based on the subset of input variables and the newly estimated weights.

Finally, the weights between the output and the last hidden layer are reestimated. Here we have two options: either we treat the output layer as a hidden layer and try to fit the incoming activity of the original network, or we can use the output targets to calculate the desired incoming activity. The first approach, which we will also take in our simulations, yields

$$
\tilde{w}_{K+1} = \arg\min_B \sum_{\mu} \| O^\mu - B \tilde{H}_{K\mu} \|^2
$$
where \( o = w_k H_k \) is the original incoming activity of the output layer, and \( \hat{H}_k \) is the activity of the last \((K^{th})\) hidden layer given the new weights and the limited input.

As already mentioned, partial retraining is far less computationally intensive than the naive approach, i.e., training a new network with the restricted input. This reduction in computational needs is accomplished by the usage of implicit knowledge about the problem which is contained in the neural network trained on all input variables. Furthermore, partial retraining has a unique solution while training a new network might converge to another (local or global) minimum.

### 3 Network Comparison

How are the performances of two networks with different sets of input variables compared? We will describe how this is done for two different tasks: a regression task and a classification task. For simplicity, we will assume that both tasks have only one output.

We assume that the performance on a regression task is measured by the mean squared error, e.g. \( E = \frac{1}{P} \sum_{\mu=1}^{P} (T^{\mu} - O^{\mu})^2 \), with \( T^{\mu} \) and \( O^{\mu} \) the target and output of pattern \( \mu \), respectively. The difference in performance between two networks is thus given by

\[
E_1 - E_2 = \frac{1}{P} \sum_{\mu=1}^{P} [2T^{\mu} - (O^{\mu}_1 + O^{\mu}_2)] [O^{\mu}_2 - O^{\mu}_1] .
\]

If the mean squared error is used, it is reasonable to assume that \( T^{\mu} - O^{\mu}_1 \) is drawn from a Gaussian distribution, and thus also the two terms between brackets in equation (1) are Gaussian distributed. Under the assumption that these two terms are independent (not correlated), the product’s average value is zero, and its standard deviation \( \sigma \) is equal to the standard deviation of the first term in brackets in equation (1) times the standard deviation of the second term. If \( P \) is large, the central limit theorem yields that the hypothesis that the performances of the two networks are identical has to be rejected with significance level \( \alpha \) if

\[
P(Z > \frac{\sqrt{P} |E_1 - E_2|}{\sigma}) < \alpha ,
\]

where \( P(Z > z) \equiv \frac{1}{2} \left( 1 - \text{erf} \left( \frac{z}{\sqrt{2}} \right) \right) \) is the probability that a value greater than \( z \) is drawn from a standard Gaussian distribution, i.e., \( N(0,1) \).

The percentage of misclassifications in a classification task is given by

\[
\frac{1}{P} \sum_{\mu=1}^{P} (1 - T^{\mu}) O^{\mu} + T^{\mu} (1 - O^{\mu}) ,
\]

where \( T^{\mu} \) and \( O^{\mu} \) are the target and output label (\( \in \{0,1\} \)) of pattern \( \mu \), respectively. The difference in misclassifications between two networks is given by

\[
C_1 - C_2 = \frac{1}{P} \sum_{\mu=1}^{P} (2T^{\mu} - 1)(O^{\mu}_2 - O^{\mu}_1) .
\]
Let us consider the hypothesis that both networks perform equally well, i.e., that both have an equal chance of misclassification $p$, which can be estimated based on the data by $p = \frac{C_1 + C_2}{2}$. Similarities between the solutions, as constructed by the networks, correlate the outputs of the networks. The ratio of simultaneous misclassifications over all misclassifications can be estimated by $q = \frac{2C_1 C_2}{C_1 + C_2}$, where $C_1/2$ denotes the percentage of simultaneous misclassifications. Given this hypothesis, we find that the term $(2T^0 - 1)(O_2^0 - O_1^0)$ to calculate the difference in misclassification is on average equal to zero, and has standard deviation of $\sigma = \sqrt{2p(1-q)}$. Again if $P$ is large, the central limit theorem yields that our hypothesis has to be rejected with significance level $\alpha$ if

$$P(Z > \frac{\sqrt{P}|C_1 - C_2|}{\sigma}) < \alpha.$$  

4 Input Selection

Since every input variable is either selected or not, we have $2^N$ possibilities for $N$ input variables. Using partial retraining to determine the performances of the network given every possible combination of input variables is therefore only feasible when the number of input variables is rather small. Alternatives, but approximations, for this brute force method are backward elimination, forward selection, and stepwise selection (see e.g. [3,7]).

Backward elimination starts with all input variables and removes the least relevant variables one at the time. Backward elimination stops if the performance of network drops below a given threshold by removal of any of the remaining input variables. Forward selection starts without any input variables, sequentially includes the most relevant variables, and stops as soon as the performance of the network exceeds a given threshold. Stepwise selection is a modified version of forward selection that permits reexamination, at every step, of the variables included in the previous steps, since a variable that was included at an early stage may be come irrelevant at a later stage when other (related) variables are also included.

In our simulations we remove the least relevant variables one at the time, but, unlike backward elimination, we do not stop when the performance of the network drops below a given threshold but we continue until all variables are removed. From these architectures, i.e., with none, one, two, ..., and $N$ inputs, the smallest architecture, for which the error is not significantly different from the architecture with the minimal error, is chosen. Note that the parameter $\alpha$ controls Occam’s razor. The smaller $\alpha$ the higher the probability that a smaller architecture is considered statistically identical to the architecture with the minimal error. In other words, the smaller $\alpha$ the higher the chance that we arrive at networks with only a few input variables.
5 Simulations

5.1 Regression

The response of our artificial regression task given the ten input variables, \(X_1, \ldots, X_{10}\) which are uniformly distributed over \([0, 1]\), is given by the following signal plus noise model [4]

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

where \(\epsilon\) is \(\mathcal{N}(0, 1)\), i.e., standard normally distributed noise. The response does not depend on the irrelevant or noisy input variables, \(X_6, X_7, X_8, X_9, \text{ and } X_{10}\). Furthermore, we assume that the ten input variables are not independent: two irrelevant inputs are identical, \(X_9 \equiv X_{10}\), as well as two relevant inputs, \(X_4 \equiv X_5\). Based on this signal plus noise model, we generated a data set of 400 input-output patterns.

We trained hundred two-layered multilayer perceptrons with ten input, seven hidden and one output unit and with the hyperbolic tangent and the identity as transfer functions of the hidden and output layer, respectively. The training procedure was as follows: starting from small random initial values, the weights were updated using backpropagation on the mean squared error of a training set of 200 randomly selected patterns out of the data set. Training was stopped at the minimum of the mean squared error of a validation set consisting of the remaining 200 patterns. With \(\alpha = 0.01\), 94 out of 100 networks choose the relevant input variables, i.e., inputs \(X_1, X_2, X_3, \text{ and } X_5\), the remaining 6 networks did not only choose the relevant input variables but also one irrelevant input. These 6 networks might have been confused by random correlations between this irrelevant input and the output in both training and validation set.

5.2 Classification

The response of our artificial classification task given the ten binary input variables, \(X_1, \ldots, X_{10}\) is given by the following rule

\[
T = X_1 \cup (X_2 \cap X_3) \cup (X_4 \cap X_5 \cap X_6),
\]

where 0 and 1 code false and true, respectively. The response thus depends on one very important variable \((X_1)\), two equally important variables \((X_2 \text{ and } X_3)\), and three less important variables \((X_4, X_5, \text{ and } X_6)\). Furthermore, the response does not depend on the four irrelevant or noisy variables, \(X_7, X_8, X_9, \text{ and } X_{10}\). Based on this rule, we generated a data set containing 400 examples.

We applied the same training procedures as in the regression task, except that the neural networks had only five instead of seven hidden units. After training, the patterns were classified based on their output. If the output of a pattern was larger than 0.5 the pattern was labeled true, otherwise the class label was false.

With \(\alpha = 0.05\), 98 out of 100 networks chose the correct architecture with inputs \(X_1, X_2, X_3, X_4, X_5, \text{ and } X_6\). The remaining two networks chose a smaller
architecture, which did not include all relevant variables. To be more precise one removed variable $X_4$, the other removed the variables $X_4$, $X_5$, and $X_6$. This might be caused by the lack of examples in the training set since only 200 patterns out of $2^{10} = 1024$ possibilities were available.

6 Discussion

In this article, we have described how input selection can be performed by using partial retraining. We focused on how the performance of neural networks with different subsets of input variables can be compared in both regression and in classification tasks. Computer simulations have shown that, in two artificial problems, this algorithm indeed selects the relevant input variables.

The algorithm described in this article can be easily generalized. For example, by viewing hidden units as input units of a smaller network [2, 10], this algorithm can also be used for hidden unit selection. Similarly, it can be used for weight pruning by estimating the relevance of a single weight.

References


