On the Selection of Hidden Neurons with Heuristic Search Strategies for Approximation

Ignacio Barrio Enrique Romero
Lluís Belanche *

Univ. Politècnica de Catalunya - Dept. de Llenguatges i Sistemes Informàtics
Barcelona - Spain

Abstract. Feature Selection techniques usually follow some search strategy to select a suitable subset from a set of features. Most neural network growing algorithms perform a search with *Forward Selection* with the objective of finding a reasonably good subset of neurons. Using this link between both fields (feature selection and neuron selection), we propose and analyze different algorithms for the construction of neural networks based on heuristic search strategies coming from the feature selection field. The results of an experimental comparison to *Forward Selection* using both synthetic and real data show that a much better approximation can be achieved, though at the expense of a higher computational cost.

1 Introduction

The main objective of feature selection in inductive learning is to select the most suitable subset from a set of features [1]. Analyzing all possible subsets is usually unattainable and a heuristic search is often carried out in order to find a good subset. One of these search strategies is *Forward Selection*, that starts with an empty subset and adds the most salient feature at a time (as measured by some cost function), keeping all the previously selected features in the current subset.

On the other hand, many neural network growing algorithms obtain the network architecture by starting from scratch and adding neurons, usually one at a time, until the network reaches a suitable performance (or begins to degenerate) [2] . Changing features into neurons, these neural network growing algorithms perform a search with *Forward Selection* with the objective of finding a good subset of neurons.

This work is devoted to approximation tasks. With the objective of improving the selection of hidden neurons, we make use of this link between feature selection and neural network growing algorithms to propose the application of heuristic feature selection search strategies to the selection of neurons.

We illustrate this idea with Sequential Approximation with Optimal Coefficients and Interacting Frequencies (*SAOCIF*) [3]. This is a growing algorithm that uses *Forward Selection* to guide the selection of hidden neurons. Different selection algorithms are proposed that change the search strategy of *SAOCIF* by other strategies from the feature selection field.

*This work is supported by Ministerio de Educación y Ciencia, under project CGL2004-04702-C02-02.

Experiments are performed with Radial Basis Funcion (RBF) networks with synthetic data and with real data from a microbiology problem. The results show that with the same number of neurons, a better approximation than *SAOCIF* is indeed achieved, though at the expense of a higher computational cost. The choice of a search strategy will hold a tradeoff between number of neurons, approximation and computational cost.

2 Background

2.1 Feature Selection

In order to find a reasonably good subset of features, many feature selection methods carry out a search process. The search process has three basic elements: a cost function which directs the search, a search strategy that decides how to continue exploring new states and an initial state where the search will start from. Two popular search strategies are:

PTA(l, r): Plus l and Take Away r. At every step, l features are added one at a time (always the one that minimizes the cost function) and then r features are removed one at a time (always the one that, after removing it, the cost function is minimized). When $l > r$, it is a growing method, and when $l < r$, it is a pruning one. *Forward Selection* is *PTA(1,0)*.

SFFS: Sequential Forward Floating Selection [4]. At every step, a feature is unconditionally added and then features are removed one at a time while the value of the cost function is better than the best value achieved until this moment with the same number of features.

2.2 Sequential Construction of Neural Networks with SAOCIF

Consider the function computed by a two layer fully connected feed-forward neural network [5] with m hidden neurons and one linear output, that can be expressed as $f_m(x) = \sum_{i=1}^{m} \lambda_i h_i(\omega_i, b_i, x)$, where $\lambda_i, b_i \in \mathbb{R}$ and usually $x, \omega_i \in \mathbb{R}^N$. We divide the weights into *output-layer weights* λ_i and *hidden-layer weights* ω_i. The biases b_i can be considered as part of the hidden-layer weights.

The optimal number of hidden units, m, is not known a priori, and it has been widely discussed in the literature. The selection of a proper number of hidden units is the aim of constructive methods. Growing methods, for example, start with a small number (usually zero) of hidden neurons and add one unit at a time until a certain stopping criterion is reached (*Forward Selection*). One of these growing methods is *SAOCIF* [3], based on the minimization of the quadratic error.

In order to select the new neuron, *SAOCIF* performs an implicit orthogonalization of the output vectors of the hidden units. The new hidden-layer weights are selected taking into account their interactions with the previously selected ones in order to minimize the sum-of-squares error (empirical risk). The interactions are discovered by means of the optimal output-layer weights.

In other words, a hidden-layer weight is considered better than another if the former allows a better approximation than the latter (after computing the optimal output-layer weights of the whole network in both cases). *SAOCIF* allows the candidate hidden-layer weights to come from different sources. When the candidate hidden-layer weights are the input points from the training set, *SAOCIF* is equivalent to Orthogonal Least Squares [6].

3 Application of feature selection search strategies to the selection of the hidden-layer weights

The main problem of *Forward Selection* is that once a feature has been added, it can no longer be removed. Therefore, the solution with N features has to include all the features from the solution with $N-1$ features.

Since *SAOCIF* uses *Forward Selection*, it may choose a neuron as being very necessary in a given moment of the training process. In a posterior moment the contribution of that neuron might be replaced, at least partially, by other neurons.

In this work we apply the aforementioned feature selection search strategies (*PTA(l,r)* and *SFFS*) to the selection of hidden neurons. In the particular case of *Forward Selection*, *SAOCIF* is obtained. Similar to the feature selection case, selecting a better subset of neurons may improve the approximation, given the same subset size.

Forward Selection is the only search strategy in Section 2.1 that does not need to remove elements. Other search strategies do include the removal of elements. In order to select the next neuron to add or remove, the implicit orthogonalization performed by *SAOCIF* is basically followed. That is, a hidden-layer weight is added/removed when its addition/removal leads to the best possible approximation.

Using more complex search strategies that minimize the sum-of-squares error, we expect two main differences of the new methods with respect to *SAOCIF*:

- Given the same network size, the approximation will tend to be better.
- They will be more costly, since they need to add and remove more neurons.

If confirmed, it would be worth to know how much the approximation is improved in relation to the computational cost.

4 Experiments

4.1 Setting

The experiments are performed with *PTA(1,0)*, *PTA(2,1)* and *SFFS*. The objective of our experiments is to check if *PTA(2,1)* and *SFFS* improve the approximation of *PTA(1,0)*, to observe how large the improvement is (if any) and what is the computational cost that we have to pay.

A preprocess is carried out with the input vectors, scaling them so that each dimension fits in the [-1,+1] range. The candidate hidden-layer weights are the examples of the training set. RBF networks with $h_i(x) = exp(-\|x-c_i\|^2/r^2)$ are used, where c_i is the RBF center (the hidden-layer weight) and r is the width. We set $r^2 = 0.3D$, following [7], where D is the dimension of the input vectors.

We use the *kin-8* and *pumadyn-8* families of data sets from the DELVE project [8]. Two versions are tested: **f**airly linear with **m**oderate noise (*fm*) and **n**on-linear with **m**oderate noise (*nm*). For each version, we use eight data sets, each with 256 training data and eight input variables.

Real data from a microbiology problem, TOFPSW [9], are also tested where the objective is tracking the origin of faecal pollution (human or animal) in surface waters. The data set consists of 103 examples with 38 input variables.

We set the maximum number of hidden neurons to 100 for the DELVE data sets and 50 for the TOFPSW one. When the method first tries to surpass that number of neurons, the algorithm is stopped.

4.2 Results

Data	Method	Rel. error	Rel. cost	Req. neurons
kin-8fm	PTA(2,1)	0.75	3	87
	SFFS	0.67	14	79
kin-8nm	PTA(2,1)	0.67	3	84
	SFFS	0.61	16	77
pumadyn-8fm	PTA(2,1)	0.68	3	82
	SFFS	0.62	16	75
pumadyn-8nm	PTA(2,1)	0.69	3	86
	SFFS	0.58	15	76
TOFPSW	PTA(2,1)	0.43	3	42
	SFFS	0.43	9	41

Table 1: Error and cost relative to *PTA(1,0)* with 100 neurons (50 neurons for the TOFPSW data set). The number of neurons required to achieve the same approximation as *PTA(1,0)* with 100 neurons (50 for TOFPSW) is also shown.

The results (average of the eight runs for the DELVE data and one run for the TOFPSW data) are sumarized in Table 1. The column 'Rel. error' shows the ratio between the error of each method in the obtained 100 neuron networks (50 for the TOFPSW data) and that of *PTA(1,0)*. The column 'Rel. cost' shows the ratio between the number of added plus removed neurons for each method with respect to *PTA(1,0)*. The column 'Req. neurons' shows the minimum number of neurons necessary to achieve at least the same approximation as *PTA(1,0)* with 100 neurons (50 for the TOFPSW data).

Figure 1 summarizes how much the approximation error of the methods is improved and how much the computational cost is worsened. It shows the means of the results obtained for the 8 runs on the *kin-8nm* data sets. The left plot shows, for each network size, the ratio between the quadratic errors on the

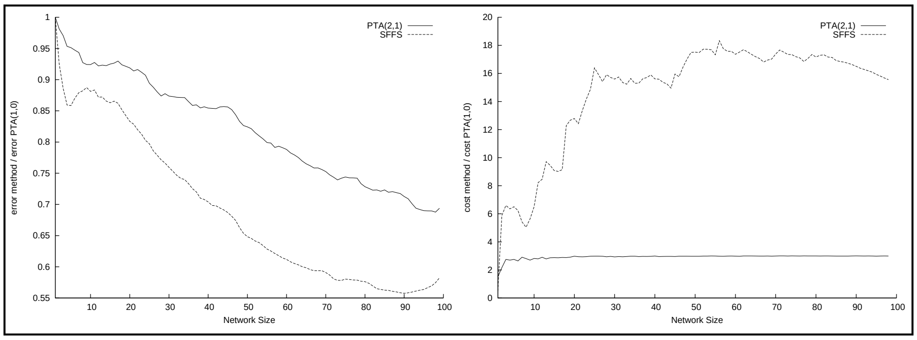

Fig. 1: The left plot shows, for each network size (measured in number of hidden neurons), the ratio between the quadratic errors on the training set obtained with each method and that obtained with *PTA(1,0)*. The right plot shows, for each network size, the ratio between the number of added plus removed neurons and the number of added neurons for *PTA(1,0)*.

training set obtained with each method and that obtained with *PTA(1,0)*. The right plot shows, for each network size, the ratio between the number of added plus removed neurons for the methods and the number of added neurons for *PTA(1,0)*.

The *SFFS* algorithm obtains the best approximation, followed by *PTA(2,1)*. The improvement over *PTA(1,0)* gets higher as the number of neurons grows. The cost of *SFFS*, with respect to that of *PTA(1,0)* also grows with the number of neurons, while the cost of *PTA(2,1)* remains the triple than that of *PTA(1,0)*.

4.3 Discussion

In the experiments, as seen in Table 1, the amount of nonlinearity of the target function does not seem to affect significantly the improvement of both methods over *PTA(1,0)*.

SFFS achieves the best approximation, at the expense of a higher computational cost. *PTA(2,1)* lies in an intermediate position between *PTA(1,0)* and *SFFS*, both in terms of computational cost and approximation.

If the objective is to finish the training as soon as possible, though at the expense of a worse approximation, we should choose *PTA(1,0)*. If the objective is to approximate as good as possible, without considering the computation time, then the method to choose is *SFFS*. In many cases, though, none of these scenarios will take place, and the decision will not be so easy.

PTA(1,0) and *PTA(2,1)* have a fixed computation time. We always know beforehand how much they will delay to achieve a solution with some given number of neurons, while with *SFFS* the computation time is not fixed, and it can delay more or less depending on the problem at hand. This is a drawback if a maximum computation time is required.

5 Conclusions and Future Work

We have tested some feature selection search strategies for the selection of hidden neurons, obtaining a much better approximation than pure *Forward Selection*, used in most network growing algorithms. *SFFS* achieves the best approximation at the expense of a much higher computational cost.

This work has focused on approximation. In many applications, neural networks are trained to achieve good generalization rather than good approximation. In some preliminary experiments we observed that, regarding the generalization performance, there is no method outperforming the others. The problem is that the search minimizes a function (training set error), but we would like to find the minimum of another function (expected risk), and although both functions come from the same problem, minimizing the sum-of-squares error may lead to overfitting. If we want to generalize well, the objective of the search should be changed to obtain a good approximation while controlling the network complexity in other ways than only the number of hidden neurons.

This problem can be tackled from a Bayesian point of view. The evidence is a measure of merit of the model, so that models that do not fit the data sufficiently well and models that overfit usually have a low evidence. Numerical evaluation of the evidence is feasible since the models used in this work have identical form to those used for Bayesian interpolation [10]. Therefore, the methods proposed here can be adapted to search for a subset of neurons that maximizes the evidence.

References

[1] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, 1998.

[2] T. Y. Kwok and D. Y. Yeung. Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[3] E. Romero and R. Alquézar. A Sequential Algorithm for Feed-forward Neural Networks with Optimal Coefficients and Interacting Frequencies. *Neurocomputing*, in press, 2006.

[4] P. Pudil, J. Novovičová, and J. Kittler. Floating Search Methods in Feature Selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press Inc., New York, 1995.

[6] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[7] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with Gaussian kernels to radial basis function classiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, 1997.

[8] C.E. Rasmussen, R.M. Neal, G.E. Hinton, D. van Camp, Z. Ghahramani, M. Revow, Kustra R., and R. Tibshirani. The DELVE Manual, 1996. www.cs.utoronto.ca/∼delve/.

[9] A. R. Blanch et al. Tracking the origin of faecal pollution in surface water. *Journal of Water and Health*, 2:249–260, 2004.

[10] D. J. C. MacKay. Bayesian Interpolation. *Neural Computation*, 4(3):415–447, 1992.