Improving the Performance of Resource Allocation Networks through Hierarchical Clustering of High – Dimensional Data*

Nicolas Tsapatsoulis[1], Manolis Wallace[1], and Stathis Kasderidis[2]

[1] School of Electrical and Computer Engineering
National Technical University of Athens
9, Iroon Polytechniou Str., 157 73 Zographou, Athens, Greece
{ntsap,wallace}@image.ntua.gr
http://image.ntua.gr/
[2] Department of Mathematics,King's College London, Strand, WC2R2LS, UK
stathis@mth.kcl.ac.uk
http://www.kcl.ac.uk/

Abstract. Adaptivity to non-stationary contexts is a very important property for intelligent systems in general, as well as to a variety of applications of knowledge based systems in era of "ambient intelligence". In this paper we present a modified Resource Allocating Network architecture that allows for online adaptation and knowledge modelling through its adaptive structure. As in any neural network system proper parameter initialization reduces training time and effort. However, in RAN architectures, proper parameter initialization also leads to compact modelling (less hidden nodes) of the process under examination, and consequently to better generalization. In the cases of high-dimensional data parameter initialization is both difficult and time consuming. In the proposed scheme a high – dimensional, unsupervised clustering method is used to properly initialize the RAN architecture. Clusters correspond to the initial nodes of RAN, while output layer weights are also extracted from the clustering procedure. The efficiency of the proposed method has been tested on several classes of publicly available data (iris, ionosphere, etc.)

1 Introduction

When functioning in an environment with non-stationary contexts both online training and adaptation are critical. Online adaptation during normal operation is a very complex problem because target outputs are not available. The problem is handled either by using reinforcement learning or semi-supervised techniques [1].

Resource Allocating Network (RAN) architectures [2], were found to be suitable for online modelling of non-stationary processes. In this sequential learning

* This work has been partially funded by the ORESTEIA IST-2000-26091/TBD project

method the network initially contains no hidden nodes. On incoming training examples, based on two criteria, either the RAN is grown, or the existing network parameters are adjusted using a least mean square gradient descent. The first criterion is based on the prediction error while the second is the novelty criterion. In the cases where hidden neurons are modelled via RBFs ,the novelty criterion states that the distance between the observation and the winning RBF neuron should be greater than a threshold. If both criteria are satisfied, then the data is memorized and a new hidden node is added to the network.

Starting from no hidden nodes is highly inefficient, since outliers in the training data may create unnecessary nodes and, therefore, increase both learning effort and convergence time and deteriorate generalization performance. Unsupervised clustering of the training data provides the means of a successful initialization of RAN architectures that initially contain RBF-type hidden nodes. Clusters can be represented through their mean vector and, either an overall spread (vector spread) or a vector of spreads, corresponding to the spread of elements in each input dimension. Clearly, such kind of parameters can be directly transferred to RBF nodes. It will be shown in subsection 2.4 that the weights connecting the hidden neurons and the output nodes of the neural network can be also easily initialized based on the clustering results.

The number of clusters that are created by an hierarchical clustering procedure does not depend solely on the data; it is also affected by the stopping criterion. In the case of RAN architectures, creating as few clusters as possible is an advantage, while in pruning techniques starting from a relatively high number of hidden nodes is not much a problem. However, in both cases, selecting the stopping criterion of data clustering is not so critical, since structured learning follows.

2 The Modified Resource Allocation Network

The RAN architecture that we adopt consists of three layers: The input layer, containing n nodes, through which an input vector $\underline{x} \in \mathcal{R}^n$ is fed to the hidden nodes, a hidden layer containing $q(t)$ RBF-type hidden nodes (at iteration t), and an output layer, containing p sigmoid nodes [3].

2.1 Learning

Learning is incorporated into the network using the gradient descent method. A squared error criterion is used as a training performance parameter. The squared error $e(t)$ at iteration t is computed in the standard way:

$$e(t) = \frac{1}{2}\sum_{k=1}^{p}(d_k(t) - y_k(t))^2$$

where $d_k(t)$ is the desired output and $y_k(t)$ is the output at node k given by:

$$y_k = \frac{1 - e^{2z_k}}{1 + e^{2z_k}}, \quad z_k = \underline{w}_k^T \cdot \underline{\phi}$$

where $\underline{w}_k = [w_{k1}\ w_{k2} \dots w_{kq(t)}]^T$, $k = \in \mathcal{N}_p$, are the weights connecting the RBF neurons with the output nodes and $\underline{\phi}$ is the output of the hidden layer.

Each hidden node represents a single RBF and computes a kernel function of $\underline{x}$ according to the following equation:

$$\phi_j(\underline{x}) = exp\{-\frac{1}{2}\sum_{i=1}^{n}(\frac{x_i - \mu_{ji}}{\sigma_{ji}})^2\}$$

where $\mu_j = [\mu_{j1}, \mu_{j2} \dots \mu_{jn}]$ and $\sigma_j = [\sigma_{j1}, \sigma_{j2} \dots \sigma_{jn}]$ are the center and the spreads of the j-th hidden node, respectively. The output of the hidden layer is given by $\underline{\phi} = [\phi_1, \phi_2 \dots \phi_n]$

The three parameters of the network (μ_j, σ_j, $j \in \mathcal{N}_{q(t)}$ and w_k, $k \in \mathcal{N}_p$) are modified on the basis of update equations taking the following forms:

$$w_{kj}(t+1) = w_{kj}(t) - \eta(t) \cdot a_j \frac{\vartheta e(t)}{\vartheta w_{kj}(t)} \tag{1}$$

$$\mu_{ji}(t+1) = \mu_{ji}(t) - \eta(t) \cdot a_j \frac{\vartheta e(t)}{\vartheta \mu_{ji}(t)} \tag{2}$$

$$\sigma_{ji}(t+1) = \sigma_{ji}(t) - \eta(t) \cdot a_j \frac{\vartheta e(t)}{\vartheta \sigma_{ji}(t)} \tag{3}$$

$\eta(t)$ is the online computed, decreasing with time, learning rate.

Parameter a_j in equations 1,2,3 is related with the j-th hidden node and accounts for soft competitive learning. In particular, a_j indicates the similarity between the j-th hidden node and the input pattern $\underline{x}(t)$, computed using the following equation:

$$a_j = 1 - \frac{\|\underline{x}(t) - \underline{\mu}_j\| - \|\underline{x}(t) - \underline{\mu}_{nearest}\|}{\|\underline{x}(t) - \underline{\mu}_{farthest}\| - \|\underline{x}(t) - \underline{\mu}_{nearest}\|}$$

where $\underline{\mu}_{farthest}$ and $\underline{\mu}_{nearest}$ are centers of the farthest and nearest hidden nodes from $\underline{x}(t)$ respectively, and $\|\cdot\|$ denotes the Euclidean distance.

2.2 Creating a hidden node

Training data are supplied to the network in the form of pairs $(\underline{x}(t), \underline{d}(t))$ of input and target vectors. If a new input $\underline{x}(t)$ does not significantly activate any hidden node and the prediction error is significantly large, a new node is created according to the following relations: $q(t) = q(t-1) + 1$, $N_{q(t)} = 1$, $\underline{\mu}_{q(t)} = \underline{x}(t)$, $\underline{\sigma}_{q(t)} = k \cdot \|\underline{x}(t) - \underline{\mu}_{nearest}\|$ and $w_{kq(t)} = d_k(t) - y_k(t), \quad k \in \mathcal{N}_p$, where k is a constant (overlap factor).

2.3 Updating the network

If the new input $\underline{x}(t)$ activates at least one of the hidden nodes, or the prediction error is small, the network parameters are updated based on equations 1,2,3 and 4, 5, 6:

$$\frac{\vartheta e(t)}{\vartheta w_{kj}(t)} = \phi_j(\underline{x}(t))\{d_k(t) - y_k(t)\}\{1 - (y_k(t))^2\} \quad (4)$$

$$\frac{\vartheta e(t)}{\vartheta \mu_{ji}(t)} = \phi_j(\underline{x}(t))\frac{\{x_i(t) - \mu_{ji}(t)\}}{\sigma_{ji}^2(t)} \sum_{k=1}^{p}(w_{kj}(t)\{d_k(t) - y_k(t)\}\{1 - (y_k(t))^2\}) \quad (5)$$

$$\frac{\vartheta e(t)}{\vartheta \sigma_{ji}(t)} = \phi_j(\underline{x}(t))\frac{\{x_i(t) - \mu_{ji}(t)\}}{\sigma_{ji}^3(t)} \sum_{k=1}^{p}(w_{kj}(t)\{d_k(t) - y_k(t)\}\{1 - (y_k(t))^2\}) \quad (6)$$

2.4 Initialization of the Network Parameters

The network is initialized by setting the values of $\underline{\mu}_j$, $\underline{\sigma}_j$, $j \in \mathcal{N}_{q(t)}$ and $\underline{w}_k$, $k \in \mathcal{N}_p$, according to the results of the hierarchical clustering algorithm. In particular, the centers $\underline{\mu}_j$ of the hidden RBF neurons are obtained directly from the centers of the created clusters, while the spreads are set according to the following equation:

$$\sigma_{ji} = \sqrt{\frac{1}{N_j}\sum_{k=1}^{N_j}(\nu_{ji}^k - m_j)^2}$$

where ν_{ji} is the i-th element of the k-th vector of the j-th cluster, $\underline{m}_j$ is the center of the cluster and N_j is the number of vectors of the cluster. Weights $\underline{w}_k$ are determined by considering the way elements of detected clusters are mapped to output classes. Specifically, if $per\%$ of the elements of cluster j belong to class k, then the corresponding hidden node is linked to the class's output node with a weight of $w_{kj} = \frac{per}{100}$.

3 Hierarchical Clustering of High – Dimensional Data

When the count of clusters that exist in a data set is not known beforehand, partitioning methods are inapplicable; an hierarchical clustering algorithm needs to be applied [4]. Their general structure is as follows:

1. Turn each input element into a singleton, i.e. into a cluster of a single element.
2. For each pair of clusters c_1, c_2 calculate a compatibility indicator $CI(c_1, c_2)$. The CI is also referred to as cluster similarity, or dissimilarity, measure.
3. Merge the pair of clusters that have the best CI. Depending on whether this is a similarity or a dissimilarity measure, the best indicator could be the maximum or the minimum operator, respectively.
4. Continue at step 2, until the termination criterion is satisfied. The termination criterion most commonly used is the definition of a threshold for the value of the best compatibility indicator.

This process creates a dendrogram of partitionings on the data.

The core of this generic algorithm is the definition of a unique compatibility indicator among any pair of clusters. When the input space has more than one dimensions, an aggregating distance function, such as Euclidean distance, is typically used as the CI. This, of course, is not always meaningful. Cases exist, in which the "context" can change the similarity or dissimilarity measure to be used. In such cases, a selection of distance function among elements needs to be performed, prior to calculating a CI among clusters.

Real elements are usually grouped together semantically, based on their similarity in a single or a few features. When the total number of features is high, small distances in a small subset of them barely affect the overall distance, when an aggregation of distances in all features is used. Thus, only when the correct subset of features is considered, can elements be compared correctly. In this paper we tackle feature selection based on the following principle: while we expect elements of a given set to have random distances from one another according to most features, we expect them to have small distances according to the features that relate them. We rely on this difference in distribution of distance values in order to identify the *context*, i.e. the features that most probably relate a set of elements.

More formally, let c_1 and c_2 be two clusters of elements. Let also r_i, $i \in \mathcal{N}_F$ be the metric that compares the i-th feature, and F the count of features (the dimension of the input space). A distance (dissimilarity) measure between the two clusters, when considering the i-th feature, is given by

$$f_i(c_1, c_2) = \sqrt[\kappa]{\frac{\sum_{a \in c_1, b \in c_2} [r_i(a_i, b_i)]^\kappa}{|c_1||c_2|}}$$

where e_i is the i-th feature of element e, $|c|$ is the cardinality of cluster c and $\kappa \in R$ is a constant.

The context is a selection of features to consider when calculating an overall distance value. We can define it as a vector $\underline{x}$ of $\mathcal{R}_F^+$, with $\sum_{i=1}^{F} x_i = 1$. Then the overall distance between c_1 and c_2 is calculated as

$$d(c_1, c_2) = \sum_{i=1}^{F} [x_i(c_1, c_2)]^\lambda f_i(c_1, c_2)$$

where $\lambda \in R$ is a constant and x_i is the degree to which f_i is included in the context.

The features that relate c_1 and c_2 are "most probably" the ones that produce the smallest distances f_i. Therefore, the "correct" context can be calculated as the context that produces the best (smallest) overall distance.

When $\lambda = 1$ the optimization is trivial: the feature that produces the smallest distance is the only one selected. The degree to which it is selected is 1. If more than one features produce the best distance, then they are equally selected, as there is no information as to which should be favored.

Table 1. Classification rates and numbers of hidden nodes

	No training	Random	Bayesian	Pre-clustering
Number of rules	3	6	5	3
Classification rate	87.33%	96%	97.3	98%

When $\lambda \neq 1$ and $f_i(c_1, c_2) \neq 0 \forall i \in \mathcal{N}_F$, then it is easy to prove that the best context is given by:

$$x_1(c_1, c_2) = \frac{1}{\sum_{i=1}^{F} [\frac{f_1(c_1,c_2)}{f_i(c_1,c_2)}]^{\frac{1}{\lambda-1}}} \quad and \quad x_i(c_1, c_2) = x_1 [\frac{f_i(c_1, c_2)}{f_1(c_1, c_2)}]^{\frac{1}{\lambda-1}}$$

where $i \in \mathcal{N}_F$. Proof is omitted for the sake of space.

When $\lambda \neq 1$ and $\exists i \in \mathcal{N}_F : f_i(c_1, c_2) = 0$, then the features for which $f_i(c_1, c_2) = 0$ are the ones the are (equally) selected.

As λ increases, pairs of clusters that are related by fewer features, and thus have greater values in their contexts, are obviously assigned greater distances. It order for distances to be used as compatibility indicators it is, of course, imperative that they are transformed as to become directly comparable to each other, even when different contexts are used for different pairs of clusters. Therefore, the following compatibility indicator is used:

$$CI(c_1, c_2) = \frac{d(c_1, c_2)}{x_\lambda(c_1, c_2)} \quad where \quad x_\lambda(c_1, c_2) = \sum_{i=1}^{F} [x_i(c_1, c_2)]^\lambda$$

As far as the termination criterion is concerned, a threshold on the growth rate of value of CI is used. In other words, when the best CI starts increasing rapidly, we conclude that all valid clusters have already been detected and are starting to be merged with each other. Therefore, the algorithm terminates.

The average values of features of a detected cluster c_j form the centroid $\underline{m_j} = [m_{j1}, m_{j1} \ldots m_{jF}]$, i.e. a "virtual" element that is located in the center of the cluster. In this equation, m_{ji} is given by $m_{ji} = \frac{\sum_{a \in c_j} a_i}{|c_j|}$

4 Experimental Results

In order to demonstrate the efficiency of the proposed scheme, we have applied it to several well known data sets. The iris data set contains 150 elements, characterized by 4 features, that belong to three classes; two of these classes are not linearly separable from each other. This is a relatively easy data set, as the number of clusters in the data is equal to the number of classes. For testing purposes, a part of the set was used as training data.

Four different experiments were carried out for the iris data set:

– The network was initialized as described in subsection 2.4, based on the results of the clustering, and was not trained.

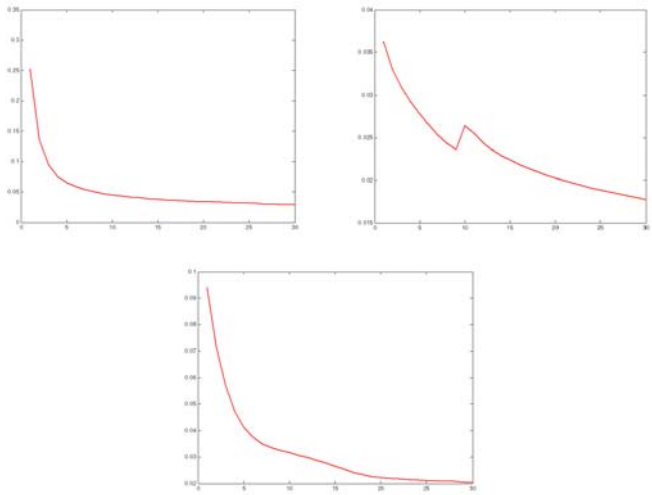

Fig. 1. MSE as a function of epochs: (a)random initialization (b) bayesian initialization (c) with pre-clustering

- The network was initialized with three random hidden nodes
- A bayesian approach was used. Specifically, the three existing classes were used as clusters, and the network was initialized as described in subsection 2.4.
- The network was initialized as described in subsection 2.4, based on the results of the clustering.

As can be seen in Figure 1, where the MSE is presented as a function of the number of epochs, random initialization has an upper bound of performance, which is probably caused by a local minimum that the network has to overcome. Properly initialized approaches, on the other hand, progress much better. The bayesian approach seems to be better, as far as MSE during the first epochs is concerned, but only the network that was initialized using the high – dimensional pre-clustering continues to improve drastically. After a few epochs (about 20), the proposed scheme has an MSE that is considerably lower that those of different approaches.

Moreover, Table 1 presents the classification rate and the number of hidden nodes in the final network. The first remark we can make is that the network that was not trained performed much worse than others, indicating that training is necessary for this data set. As far as the classification rate and the number of rules are concerned, our method converges to a network of just three rules, no more than the classes of the problem, while not loosing in performance when compared to networks with larger numbers of hidden nodes. In addition to the experiments presented herein, our method outperforms others in the literature

as well (7 rules, 96.7% [5]) (17 rules, 95.3% [6]) (9 rules, 95.3% [7]) (7 rules, 96% [8]).

Similar results are observer for other data sets as well. Especially for the ionosphere data set, which is clearly a high dimensional data set (it is characterized by 34 features), simpler initialization approaches, such as the bayesian approach, have proven to be totally ineffective.

5 Conclusions

In this paper, we combine an hierarchical clustering algorithm with a modified Resource Allocation Network in order to properly initialize the network parameters and especially the RBF nodes of the hidden layer. RANs are dynamically formed architectures and, thus, provide the means to model non-stationary phenomena. On the other hand, proper initialization serves two purposes: (a) Reduces the learning effort, (b) keeps the number of hidden nodes low and increases generalization performance; this is highly desirable since in RBF networks one may consider the hidden nodes to be "rules" and therefore use the RAN architecture for knowledge extraction from numerical data [9]. The latter is very important since there are several domains in which no estimation about the number of rules that are required to solve a particular problem is available. Moreover, "rules" can be created to model a changing context, given the dynamic nature of RANs.

The classification performance of the proposed network turns out to be excellent. When initiated with three hidden nodes, based on the results of the high – dimensional clustering, outperforms the majority of the soft-computing schemes that were tested on the iris classification problem. It performs similarly in other data sets as well, especially as the dimensionality of the input space increases.

References

1. Vapnik, V.: Statistical Learning Theory. John Willey and sons (1998)
2. Platt, J.: A resource-allocating network for function interpolation. Neural Computing **3**, (1991) 213-225
3. Lee, K., Street, W. N.: Intelligent Image Analysis using adaptive resource allocating networks. Procs of IEEE International Workshop on NN for Signal Processing (2001)
4. Theodoridis, S., Koutroumbas, K.: Pattern Recognition, Academic Press (1998)
5. Nauk, D., Kruse, R.: A neuro-fuzzy method to learn fuzzy classification rules from data. Fuzzy sets and Systems **8** (1997) 277-288
6. Kasabov, N., Woodford, B.: Rule insertion and rule extraction from evolving fuzzy neural networks: Algorithms and applications for building adaptive, intelligent, expert systems. Procs of FUZZ-IEEE'99, (1999) 1406-1411
7. Kasabov, N.: Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. Fuzzy Sets and Systems **82**, (1996) 135-149
8. Halgamuge, S., Glesner, M.: Neural Networks in designing fuzzy systems for real world applications. Fuzzy Sets and Systems **65**, (1994) 1-12
9. Mitra, S., DE, R.K., Pal, S.K.: knowledge-based fuzzy MLP for classification and rule generation. IEEE Trans. Neural Networks **8**,(1997) 1338-1350