A LEARNING MODEL FOR INTELLIGENT AGENTS USING RADIAL BASIS FUNCTION NEURAL NETWORKS WITH ADAPTIVE TRAINING METHODS

BY

IRINA DIACONIȚA and FLORIN LEON

“Gheorghe Asachi” Technical University of Iași, Faculty of Automatic Control and Computer Engineering

Received: March 7, 2011
Accepted for publication: June 3, 2011

Abstract. Learning is an important ability for the agents, as it increases their flexibility and adaptability. Inductive learning can be a way for intelligent agents to automatically discover knowledge in large datasets. In this paper we describe a learning model based on radial basis function (RBF) neural network and we study adaptive training methods, which are particularly useful for datasets with many training instances, similar to the situations encountered by intelligent agents acting in complex, open environments.

Key words: learning model, radial basis functions, neural networks, intelligent agents, classification, regression.

2000 Mathematics Subject Classification: 68T05, 68T42.

1. Introduction

Learning is an important ability for the agents, as it increases their flexibility and adaptability. For adaptive and deliberative agents, learning enables multiagent systems to be more flexible and robust, and it makes them
better able to deal with uncertainty and changing conditions, which are key factors for intelligent behaviour.

A learning agent can adapt to its user’s preferences or it can decide which agents to trust and cooperate with, and which ones to avoid. It can recognize previously encountered situations and improve its performance based on its experience. Inductive learning can be a way for intelligent agents to automatically discover knowledge rather than having it predefined (Seydim, 1999).

In this paper, we describe a learning model for intelligent agents based on radial basis function (RBF) neural network and we study adaptive training methods, different from the standards techniques which involve solving (large) equation systems. These adaptive methods are particularly useful for datasets with many training instances, and therefore they are fit for the situations encountered by intelligent agents acting in complex, open environments.

2. Radial Basis Functions Neural Networks

The learning process of neural networks can be seen as a multidimensional curve fitting problem, i.e. finding a surface in a multidimensional space to fit the surface described by the input data.

The topology of a RBF neural network is presented in Fig. 1. The input of a neuron $x$ from the middle layer is a function of the distance between the centres and the input vector $t$: $r = \|t - x\|$. A common activation function is $\varphi(r) = e^{-\beta r^2}$, with $\beta > 0$. It is obvious that the value of the radial function is maximum if the input (the distance) is 0. If the distance between $t$ and $x$ increases, the value of the output decreases. Therefore, a neuron behaves like a detector which produces 1 every time the input pattern is equal to the weight vector $t$.

![Fig. 1 – The topology of a RBF network.](image)
The Gaussian function is often used to compute the output of the middle layer neuron, and in this case the output of the network for an input vector \( x \) and \( N \) centres is:

\[
f(x) = \sum_{j=1}^{N} w_j \exp\left( -\frac{1}{2\sigma^2} \|x - x^j\|^2 \right)
\]  

(1)

2.1. A Comparison Between RBFs and Multilayer Perceptrons

Both the multilayer perceptron (MLP) and the RBF networks are composed of an input layer, which has the role of distributing the data, hidden layer(s) and an output layer. In both cases there are no feedback connections, but only forward connections between layers. Both types of networks have, in many cases, linear neurons in the output layer, and generally the non-linear computing units are located in the hidden layer(s). Both the multilayer perceptron and the RBF networks, can be trained using a supervised algorithm.

MLP networks may have two or more hidden layers of non-linear, structurally identical neurons. RBF networks have only one hidden layer of neurons with radial activation functions. While in the case of MLP networks, the output of a hidden neuron is a nonlinear function of the weighted sum of its inputs, in the case of RBF networks a hidden neuron computes the distance (the Euclidean norm) between itself and the input vector. Therefore, the functioning principles of the RBF and MLP neurons are totally different (Stergious & Siganos, 1996).

The multilayer perceptron is an universal approximator. If it is trained on a consistent dataset, it will be able to approximate the value of the function for any given input. A RBF networks is an universal interpolator, making a local approximation of the given function. Its output will approximate the value of the function in the interpolation centres, but its precision decreases exponentially as the points are farther from the centres. The way the approximation is done depends on the value of the variance (\( \sigma^2 \)). The smaller \( \sigma \) is, the greater is the decrease in performance as the distance from the centre grows. By increasing the value of \( \sigma \), the network apparently becomes an universal approximator, but it becomes insensitive to local changes in the shape of the curve. Due to the local approximator property of the RBF networks, the training algorithms for them converge faster (Ekeberg & Herman, 2006; Hill & Lewicki, 2007; Orr, 2000).

3. Algorithms for Centre Selection

If we use all data instances as centres, the precision of the results is very good, but for is large datasets the computation process can become very
complex. For this reason it is recommended to pick a number of centres from the input data. There are a few different methods to choose these centres: random selection, unsupervised selection (clustering), gradient descent, etc. When selecting the centres randomly, there is no control of the relevance of the centres and the approximation errors, which can vary a lot from one experiment to another. In what follows, we will present some algorithms that can be used to select the centres in a more meaningful way.

3.1. Clustering

Clustering can be considered the most important unsupervised learning problem. Its aim is to organizing objects into groups (clusters) such that the members of a group are more similar to one another than to members of the other groups.

K-means (MacQueen, 1967) is one of the most popular unsupervised learning algorithms for clustering. The main idea is to define $k$ centroids, one for each cluster. Although it can be proved that the procedure always converges, it does not always find the optimal configuration. Therefore, the algorithm is also very sensitive to the positions of the initial centres, which are chosen randomly and it can be applied several times in order to reduce this effect.

3.2. Cross-Validation

Cross validation is a statistical method of evaluation and comparison for learning algorithms, by splitting the data into two parts: one for learning or training a model and one for the validation of the model. Usually, the training and validation sets must be crossed in a number of iterations, so that each element is validated by all the other elements. K-fold cross validation is the basic type of cross validation and all the other types are special cases or involve applying it more times. (Refaeilyadeh et al., 2008).

In $k$-fold cross validation, the data is first split in $k$ (approximately) equal groups and then $k$ training and validating iterations are executed, so that at each step one of the $k$ groups is used for training and the remaining $k-1$ groups for validation/testing. In the end, there will be $k$ samples of performance measurements for each algorithm. There are different methods to get aggregate results from these samples, such as computing the mean values. These samples can also be used to test some hypotheses in a statistic manner, in order to prove that one algorithm is superior to another on a given problem.

Cross validation is, at the moment, one of the best ways to estimate the generalization capacity of an algorithm.
3.3. The Gradient Descent Algorithm

The gradient descent rule adapts the weights with an amount proportional to the first derivative (the gradient) of the error of a processing unit, based on the current weight. Its aim is to decrease the error function, to avoid its local optima and to reach its global optimum.

For architectures with \( i \) output neurons, the weight adjustment can be done iteratively, by a rule of error minimization. The most common method is to reduce the gradient of the error function step by step, which, like in the case of MLP, provides an equation for weight adjustment at each iteration (Bayen, 2005):

\[
w_{mk}(j+1) = w_{mk}(j) - \eta \frac{\partial E(j)}{\partial w_{mk}}(j), \quad j = 1, 2, \ldots, N
\]

(2)

where: \( w_{mk} \) is the weight of the connection between the \( k \)th neuron from the hidden layer and the \( m \)th neuron of the output layer, and \( j \) is the current iteration (Moore, 2003).

Thus, iterative training for centre selection is applied using the formulas presented below. If \( \Delta w_y \) is the weight adjustment and \( \Delta \mu_j \) the centre position adjustment, then:

\[
\Delta w_y = \eta_1 \left( y^{(k)}_i - f_i \left( x^{(k)} \right) \right) \phi_j \left( x^{(k)} \right)
\]

(3)

\[
\Delta \mu_j = \eta_2 \phi_j \left( x^{(k)} \right) \left\| x - \mu_j \right\| \sum_{i=1}^{m} w_{yi} \left( y^{(k)}_i - f_i \left( x^{(k)} \right) \right)
\]

(4)

\[
\Delta \sigma_j = \eta_3 \phi_j \left( x^{(k)} \right) \left\| x - \mu_j \right\|^2 \sum_{i=1}^{m} w_{yi} \left( y^{(k)}_i - f_i \left( x^{(k)} \right) \right)
\]

(5)

Finally:

\[
w_y \leftarrow w_y + \Delta w_y
\]

(6)

\[
\mu_j \leftarrow \mu_j + \Delta \mu_j
\]

(7)

\[
\sigma_j \leftarrow \sigma_j + \Delta \sigma_j
\]

(8)
In the following subsections, some experiments are presented where the RBF model was applied on benchmark problems for regression and classification.

4.1. Regression Problems

Two Advanced Very High Resolution Radiometer (AVHRR) images of a cloudy sky (Frank & Asuncion, 2010) were analysed, one in the visible and the other in the infrared spectrum. Each pixel is divided in superpixels 16 x 16 and the values of the following parameters are computed: for the visible spectrum: the mean, maximum, minimum, mean distribution, contrast, second angular momentum and for the infrared spectrum: the mean, maximum and minimum. We used RBF neural networks for approximating the entropy.

The regression with centres in all input instances has a mean absolute error of 0.0000 and the learning process takes 101.8594 sec.

In Table 1, the errors and execution times for regression with randomly chosen centres are presented. The results vary a lot from one experiment to another, but it can be noticed that when the number of centres increase, the errors vary less, and have smaller values. Obviously, choosing a larger number of centres leads to an increased execution time. The tests were performed on a computer with an Intel Core 2 Duo (T7300, 2 GHz) processor and 2 GB of RAM.

<table>
<thead>
<tr>
<th>No. of centres (%) of the input data</th>
<th>150 (14.6%)</th>
<th>358 (35%)</th>
<th>563 (55%)</th>
<th>768 (75%)</th>
<th>921 (90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean absolute error</td>
<td>0.0081</td>
<td>0.0076</td>
<td>0.004</td>
<td>0.002</td>
<td>0.0009</td>
</tr>
<tr>
<td>Execution time (s)</td>
<td>2.2556</td>
<td>10.9531</td>
<td>28.5781</td>
<td>57.5156</td>
<td>86.5156</td>
</tr>
</tbody>
</table>

When selecting the centres by k-means clustering, we see that the precision of the results is much better than when selecting the centres randomly, even for a small number of centres (14%), but the computation time is longer, as shown in Table 2. As we increase the number of centres, the difference between the mean absolute errors decreases. For a large (percentage-wise) numbers of centres, the difference between the execution times of the two algorithm becomes negligible due to the fact that the selection of the centres and the distribution of instances is the most time consuming step of clustering. For a large number of centres (over 60%), their position and the elements in their groups do not vary too much, resulting in a
shorter processing time. Finally, choosing 90% of the input data instances leads to equal errors and execution times regardless of the way they are selected (randomly or by k-means clustering). This confirms the utility of k-means clustering for a small number of centres and also the fact that, for a large(r) number of centres, the algorithms used to select them are less important. These cases are close to that of the regression with centres in all instances, but the computation times are still better.

In the case of k-means clustering the results vary from one experiment to another as well, because the initial centres are chosen randomly, but the differences are small, because the computation errors have to stay under a certain threshold.

Table 2

The Results of the Regression with Centres Chosen by k-Means Clustering

<table>
<thead>
<tr>
<th>No. of centres (%) of the input data</th>
<th>150 (14.6%)</th>
<th>358 (35%)</th>
<th>563 (55%)</th>
<th>768 (75%)</th>
<th>921 (90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean absolute error</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0016</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Execution time (s)</td>
<td>38.571</td>
<td>113.3438</td>
<td>31.5313</td>
<td>59.3594</td>
<td>88.8594</td>
</tr>
</tbody>
</table>

When choosing the centres through cross validation, the number of centres is computed by the agent itself. When using k-means clustering, the selection of the centres was influenced by the centres and thus the results varied even for the same number of centres. Cross validation will always select the same centres, therefore the precision and computation time will always be the same, unless we change the threshold of the acceptable error or the maximum number of iterations. In our case, there is a mean absolute error of 0.0281, for only 3 centres (0.29% of the input data instances). The computation time is lower than that for clustering, ranging between 28 and 31 sec.

When selecting the centres by the gradient descent method, both the precisions and the execution times vary depending on the learning rates and the number of centres. Table 3 presents the evolution of the absolute mean error for different numbers of centres and different values of the learning rates. Table 4 presents the evolution of the computation times depending on the same parameters.

It can be noticed that the precision and computation time become better when the learning rates decrease and the number of centres increase. However, even for a small number of centres, we can still obtain errors that are close to those of the regression with a large number of centres and much better execution times by decreasing the learning rates.
Table 3  
*The Mean Absolute Errors of the Regression with Centres Selected Using the Gradient Descent Algorithm*

<table>
<thead>
<tr>
<th>No. of centres (% of the input data)</th>
<th>150 (14.6%)</th>
<th>358 (35%)</th>
<th>563 (55%)</th>
<th>768 (75%)</th>
<th>921 (90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>µ1 = 0.8  µ2 = 0.8  µ3 = 0.8</td>
<td>0.0028</td>
<td>0.0003</td>
<td>0.0014</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>µ1 = 0.5  µ2 = 0.5  µ3 = 0.3</td>
<td>0.0002</td>
<td>0.0013</td>
<td>0.0000</td>
<td>0.0013</td>
<td>0.0000</td>
</tr>
<tr>
<td>µ1 = 0.1  µ2 = 0.3  µ3 = 0.3</td>
<td>0.0023</td>
<td>0.0004</td>
<td>0.0000</td>
<td>0.0006</td>
<td>0.0006</td>
</tr>
<tr>
<td>µ1 = 0.1  µ2 = 0.1  µ3 = 0.1</td>
<td>0.0080</td>
<td>0.0003</td>
<td>0.0014</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Table 4  
*The Computation Times of the Regression with Centres Selected Using the Gradient Descent Algorithm*

<table>
<thead>
<tr>
<th>No. of centres (% of the input data)</th>
<th>150 (14.6%)</th>
<th>358 (35%)</th>
<th>563 (55%)</th>
<th>768 (75%)</th>
<th>921 (90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>µ1 = 0.8  µ2 = 0.8  µ3 = 0.8</td>
<td>6.4219</td>
<td>13.2969</td>
<td>34.2344</td>
<td>64.8906</td>
<td>82.5938</td>
</tr>
<tr>
<td>µ1 = 0.5  µ2 = 0.5  µ3 = 0.3</td>
<td>5.2500</td>
<td>13.4531</td>
<td>67.0625</td>
<td>64.7031</td>
<td>82.1250</td>
</tr>
<tr>
<td>µ1 = 0.3  µ2 = 0.3  µ3 = 0.1</td>
<td>5.2812</td>
<td>13.5156</td>
<td>33.6875</td>
<td>65.1250</td>
<td>96.5000</td>
</tr>
<tr>
<td>µ1 = 0.1  µ2 = 0.1  µ3 = 0.1</td>
<td>2.9062</td>
<td>13.2944</td>
<td>34.8438</td>
<td>64.3125</td>
<td>95.1719</td>
</tr>
</tbody>
</table>

Fig. 2 shows the results of the regression with centres chosen by gradient descent with 250 centres and learning rates of 0.1.
Fig. 2 – The results of the regression with centres chosen by gradient descent with 250 centres and learning rates of 0.1.

Therefore, by using cross validation, the results are very good for a very low number of centres (0.29% for the input data) and, furthermore, the results do not vary from one experiment to another. When selecting more centres (15 - 35%), the results of gradient descent are comparable to those of cross validation, but the computation times are much better. K-means clustering, despite having longer execution times than cross validation, still provides the smallest mean absolute errors.

### 4.2. Classification Problems

The procedure we chose for solving classification problems is “one-versus-all” (Hastie et al., 2009). A RBF network was built for regression for every combination of any two classes, the elements from first class being assigned a 0 output value and the elements from the second a 1 output value. Each networks casts a vote regarding the class the input instance belongs to. The instance is assigned to the class with the largest number of votes.

We consider for our case study the problem of the classification of Iris flowers (Fisher, 1936). The dataset is composed of 150 instances, belonging to three different classes, which represent the three types of Iris flowers. One of the classes is linearly separable from the other two; the latter two are not linearly separable one from another. Each instance has four attributes: the length and width of the sepals and petals.
When using the classification with centres in all data instances, the results have a very good precision, but the computation time is higher than for other methods of centre selection, such as k-means clustering. The absolute mean value of the error is 0 for an execution time of about 0.42 – 0.48 sec. By choosing the centres randomly, we can notice a decrease in precision, as the centres are irrelevant most of the time for the given dataset.

In the case of k-means clustering, we obtained an absolute mean error of 0 by using 30-60% of the input instances as centres (Fig. 3). This confirms the efficiency of k-means clustering for a small number of centres. The errors are very small and the computation time is close to that of the classification with randomly chosen centres.

![Fig. 3 – The results of the classification with the centres selected by k-means clustering.](image)

When selecting the centres with cross validation, the number of the centres is automatically computed. In the case of k-means clustering, the selection of the centres also depends on the initial centres and thus one can get different precisions even for the same number of centres. When using cross validation, the centres, and therefore the output and error are always the same. They can change only if we change the threshold for the acceptable error or the maximum number of iterations. In this case, the error is 0.0133, for an execution time of 1.0781 sec.

When using the gradient descent algorithm to select the centres, both the precision and the computation times vary depending on the values of the learning rates and the number of centres.

It should be noted that the precision and the execution times increase for smaller learning rates and larger numbers of centres. On the given dataset,
absolute mean errors of 0 are obtained in 0.45 - 0.5 sec. An increase in the
number of centres and a decrease in the learning rates leads to a longer
computation time. Usually, absolute mean errors of 0 are obtained either by
using 80% of the input data as centres, for learning rates over 0.1 or by having
learning rates of 0.1 and using about 40% of the input data as centres.

Therefore, k-means clustering offers very good results for a small
number of centres. Cross validation always provides the same results and a very
good precision, but is time consuming. The gradient descent algorithm has good
results for small learning rates, but is more time consuming than k-means
clustering, for the same precision.

5. Conclusions

Compared to MLP neural networks, the RBF neural networks require
more processing units, but their training is less time consuming than that of
MLPs. This happens due to the fact that the output of the hidden neurons of a
MLP network are responsible for large areas of the input space, while the
neurons of a RBF network are responsible for relatively small regions of the
input space. As a consequence, RBF networks behave better when there are
more training instances available.

When using all input data as centres for RBF, the results are very good,
but the computations are complex and time consuming. When selecting the centres
by cross validation, the precision is quite good, even for a very small number of
centres, but the computation time is still quite high. For a small or average number
of centres (20 - 60% of the given dataset), k-means clustering and the gradient
descent algorithm (with learning rates close to 0.1) have both very good precisions
and computation times (the results are similar for the two algorithms).

These adaptive methods for training RBF networks can be used to
design the learning model of an intelligent agent.

Acknowledgements. This work was supported by CNCSIS-UEFISCUSU,
project number PNII-IDEI 316/2008, Behavioural Patterns Library for Intelligent
Agents Used in Engineering and Management.

REFERENCES

Eugenics, 7, Part. II, 179–188, 1936.
MODEL DE ÎNVĂȚARE PENTRU AGENȚI INTELIGENȚI FOLOSIND REȚELE NEURONALE CU FuncȚII DE BAZĂ RADIALĂ CU METODE DE ANTRENARE ADAPTIVE

(Rezumat)

Învățarea este o caracteristică importantă pentru agenți, crescându-le flexibilitatea și adaptabilitatea. Învățarea inductivă este o modalitate prin care agenții inteligenți pot descoperi cunoștințe în mulțimi mari de date. În această lucrare este descris un model de învățare bazat pe rețele neuronale cu funcții de bază radială și sunt studiate metode de antrenare adaptivă, care sunt utile în mod special pentru mulțimi de date cu mulțe instanțe de antrenare, caz similar situațiilor întâlnite de agenții inteligenți care rulează în medii de execuție complexe și deschise.