An Experimental Study on Training Radial Basis Functions by Gradient Descent

Joaquín Torres-Sospedra, Carlos Hernández-Espinosa,
and Mercedes Fernández-Redondo

Departamento de Ingenieria y Ciencia de los Computadores, Universitat Jaume I,
Avda. Sos Baynat s/n, C.P. 12071, Castellon, Spain
{jtorres, espinosa, redondo}@icc.uji.es

Abstract. In this paper, we present experiments comparing different training algorithms for Radial Basis Functions (RBF) neural networks. In particular we compare the classical training which consist of an unsupervised training of centers followed by a supervised training of the weights at the output, with the full supervised training by gradient descent proposed recently in same papers. We conclude that a fully supervised training performs generally better. We also compare *Batch training* with *Online training* and we conclude that *Online training* suppose a reduction in the number of iterations.

1 Introduction

A RBF has two layer of neurons. The first one, in its usual form, is composed of neurons with Gaussian transfer functions (GF) and the second has neurons with linear transfer functions.

The output of a RBF can be calculated with equations (1) and (2).

$$\hat{y}_{i,k} = \mathbf{w}_i^T \cdot \mathbf{h}_k = \sum_{j=1}^{c} w_{ij} \cdot h_{j,k} \tag{1}$$

$$h_{j,k} = \exp\left(-\frac{\|\mathbf{x}_k - \mathbf{v}_j\|^2}{\sigma^2}\right) \tag{2}$$

Where $\mathbf{v}_j$ are the center of the Gaussian transfer functions, σ control the width of the Gaussian transfer functions and $\mathbf{w}_i$ are the weights among the Gaussian units (GU) and the output units.

As (1) and (2) show, there are three elements to design in the neural network: the centers and the widths of the Gaussian units and the linear weights among the Gaussian units and output units.

There are two procedures to design the network. One is to train the networks in two steps. First we find the centers and widths by using some unsupervised clustering algorithm and after that we train the weights among hidden and output units by a supervised algorithm. This process is usually fast [1-4].

The second procedure is to train the centers and weights in a full supervised fashion, similar to the algorithm Backpropagation (BP) for Multilayer Feedforward. This

F. Schwenker and S. Marinai (Eds.): ANNPR 2006, LNAI 4087, pp. 81–92, 2006.

procedure has the same drawbacks of Backpropagation, long training time and high computational cost. However, it has received quite attention recently [5-6].

In [5-6] it is used a sensitivity analysis to show that the traditional Gaussian unit (called "*exponential generator function*") of the RBF network has low sensitivity for gradient descent training for a wide range of values of the widths. As an alternative two different transfer functions are proposed. They are called in the papers "*lineal generator function*" and "*cosine generator function*". Unfortunately, the experiments shown in the papers are performed with only two databases and the RBF networks are compared with equal number of Gaussian unit.

In contrast, in this paper we present more complete experiments with nine databases from the *UCI Repository*, and include in the experiments four traditional unsupervised training algorithms and a fully gradient descent training with the three transfer functions analysed in papers [5-6].

Furthermore, we also presents experiments with *Batch* and *Online learning*, in the original references the training was performed in *Batch* mode and we show that Online Traning is the best alternative under the point of view of training speed.

2 Theory

2.1 Training by Gradient Descent

"Exponential (EXP) Generator" Function. This RBF has the usual Gaussian transfer function described in (1) and (2). The equation for adapting the weights by gradient descent is in (3).

$$\Delta \mathbf{w}_p = \eta \cdot \sum_{k=1}^{M} \varepsilon_{p,k}^0 \cdot \mathbf{h}_k \tag{3}$$

Where η is the learning rate, M the number of training patterns and $\varepsilon_{p,k}^0$ is the output error, the difference between target and output as in equation (4).

$$\epsilon_{p,k}^o = t_{p,k} - o_{p,k} \tag{4}$$

The equation for adapting the centers by gradient descent is the following:

$$\Delta v_q = \eta \cdot \sum_{k=1}^{M} \varepsilon_{p,k}^h \cdot (\mathbf{x}_k - \mathbf{v}_q) \tag{5}$$

Where η is the learning rate and $\varepsilon_{p,k}^h$ is the hidden error given by equation (6) and (7).

$$\varepsilon_{p,k}^h = \alpha_{q,k} \cdot \sum_{i=1}^{n_o} \varepsilon_{i,k}^0 \cdot w_{iq} \tag{6}$$

$$\alpha_{q,k} = \frac{2}{\sigma^2} \cdot \exp\left(-\frac{\|\mathbf{x}_k - \mathbf{v}_q\|^2}{\sigma^2} \right) \tag{7}$$

In the above equations n_o is the number of outputs and these equation are for *Batch training*, i.e., we adapt the variables of the network after the presentation of all the patterns of the training set.

The equations for *Online training* are basically the same, we only have to omit the sum for k=1, to M in the expressions.

For example, equation (5) in the *Online training* would be the following:

$$\Delta v_q = \eta \cdot \varepsilon^h_{p,k} \cdot (\mathbf{x}_k - \mathbf{v}_q) \tag{8}$$

And we would have to adapt v_q after each presentation of a training pattern to the network.

"Linear (LIN) Generator" Function. In this case the transfer function of the hidden units is the following:

$$h_{j,k} = \left(\frac{1}{\|\mathbf{x}_k - \mathbf{v}_j\|^2 + \gamma^2} \right)^{\frac{1}{m-1}} \tag{9}$$

Where we have used $m = 3$ in our experiments and γ is a parameter that should be determined by trial and error and cross-validation.

The above equations (3), (4), (5) and (6) are the same, but in this case $\alpha_{q,k}$ is different and is given in (10).

$$\alpha_{q,k} = \frac{2}{m-1} \cdot \left(\|\mathbf{x}_k - \mathbf{v}_q\|^2 + \gamma^2 \right)^{\frac{m}{1-m}} \tag{10}$$

"Cosine (COS) Generator" Function. In this case the transfer function is the following:

$$h_{j,k} = \frac{a_j}{\left(\|\mathbf{x}_k - \mathbf{v}_j\|^2 + a_j^2 \right)^{1/2}} \tag{11}$$

Equations (3), (4) and (5) are the same, but in this case the hidden error is different as in equation (12).

$$\varepsilon^h_{p,k} = \left(\frac{h^3_{j,k}}{a_j^2} \right) \cdot \sum_{i=1}^{no} \varepsilon^0_{i,k} \cdot w_{iq} \tag{12}$$

The parameter a_j is also adapted during training, the equation is (13).

$$\Delta a_j = \left(\frac{\eta}{a_j} \right) \cdot \sum_{i=1}^{no} h_{j,k} \cdot (1 - h^2_{j,k}) \cdot \varepsilon^h_{p,k} \tag{13}$$

2.2 Training by Unsupervised Clustering

Algorithm 1. This training algorithm is the simplest one. It was proposed in [1]. It uses adaptive k-means clustering to find the centers of the gaussian units. The process is

iterative, we successively present an input pattern and after each presentation we find the closest center and adapt this center toward the input pattern, according to equation (14).

$$c(n+1) = c(n) + \eta \cdot (x - c(n)) \tag{14}$$

Where x is the input pattern, c is the closest center and η the adaptation step.

After finding the centers, we should calculate the widths of the Gaussian units. For that, it is used a simple heuristic; we calculate the mean distance between one center and one of the closest neighbors, P, for example, the first closest neighbor ($P = 1$), the second ($P = 2$), the third ($P = 3$) or the fourth ($P = 4$).

We need a quite important trial and error procedure to design the network because the number of centers of k-means clustering should be fixed a priori and also the value P for the heuristic of the widths. In our experiments we have tried all the combinations of the followings number of centers and widths. For the centers the values 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 and 110 and for the widths we have used $P = 1$, 2, 3, 4.

Algorithm 2. This algorithm is proposed in reference [2]. However, we have slightly modified the algorithm. In the original reference it is used a truncation for the Gaussian functions and non-RBF functions in the hidden layer. We have applied the algorithm without truncation in the Gaussian functions and with only RBF units in the hidden layer.

Basically the algorithm is the following. The Gaussian units are generated incrementally, in stages k, by random clustering. Let $k = 1, 2, 3, \cdots$ denote a stage of this process. A stage is characterized by a parameter δ_k that specifies the maximum radius for the hypersphere that includes the random cluster of points that is to define the Gaussian unit; this parameter is successively reduced in every stage k ($\delta_k = \alpha \cdot \delta_{k-1}$, with α in the range 0.5-0.8). The Gaussian units at any stage k are randomly selected in the following way. Randomly select an input vector x_i from the training set I and search for all other training vectors within the δ_k neighborhood of x_i. The training vector are used to define the Gaussian unit Q (the mean C_Q is the center, and the standard deviation w_Q the width) and then removed from the training set, forming what is called the remaining training set R. To define the next Gaussian unit $Q + 1$ another input vector x_i is randomly selected from R and the process repeated. This process of randomly picking an input vector x_i is repeated until the remaining training set R is empty. Furthermore, when the number of points in the cluster N_j is less than a certain parameter β no Gaussian unit is created. The stages are repeated until the cross-validation error increases.

The algorithm is described in procedure 1. Where, ρ is the standard deviation of the distances of the training points from their centroid, γ is a lower limit for the length of the neighborhood δ_k (a parameter), TRE_k is the training set error at stage k and TSE_k is the cross-validation set error.

Algorithm 3. It is proposed in reference [3]. They use a one pass algorithm called *APC-III*, clustering the patterns class by class instead of the entire patterns at the same time. The *APC-III* algorithms uses a constant radius to create the clusters. In the reference this radius is calculated as the mean minimum distance between training patterns multiplied by a constant α , see equation (15).

$$R_0 = \alpha \cdot \frac{1}{P} \cdot \sum_{i=1}^{P} \min_{i \neq j} (\|x_i - x_j\|) \tag{15}$$

Procedure 1. Algorithm 2

1. Initialize counters and constants: $k = 0, Q = 0, \delta_1 = \rho, \alpha =$ some fraction between 0.5 and 0.8.
2. Increment stage counter $k = k + 1$. Reduce neighborhood radius: if $k > 1, delta_k = \alpha \cdot delta_{k-1}$. If $k < \gamma$ stop.
3. Select Gaussian units for the k-th stage: $j = 0, R = I$.
   (a) Set $j = j + 1$.
   (b) Select an input vector x_i at random from R, the remaining training set.
   (c) Search for all vectors in R within the δ_k neighborhood of x_i. Let this set of vectors be V_j.
   (d) Remove the set V_j from $R : R = R - V_j$. If $N_j < \beta$, go to (f).
   (e) Increment Gaussian counter: $Q = Q + 1$. Compute the center C_Q and width w_Q of the Q-th Gaussian unit: $C_Q =$ centroid of the set V_j, and $w_Q =$ standard deviation of the points in the random cluster V_j.
   (f) If R is not empty, go to (a), else go to (4).
4. Calculate the RBF neural network output weights with Q number of Gaussian units.
5. Compute TSE_k and TRE_k.
   (a) If $TSE_k < TSE_{k-1}$, go to 2).
   (b) If $TSE_k > TSE_{k-1}$ and $TRE_k > TRE_{k-1}$, go to 2).
   (c) Otherwise, stop. Overfitting has occurred. Use the net generated in the previous stage.

The algorithm is described in procedure 2. The widths are calculated with the following heuristic: find the distance to the center of the nearest cluster which belongs to a different class and assign this value multiplied by β to the width.

Procedure 2. Algortihm 3

1. Select one input pattern and construct the first cluster with center equal to this pattern.
2. Repeat steps 3 to 5 for each pattern.
3. Repeat step 4 for each cluster.
4. If the distance between the pattern and the clusters is less than R_0 include the pattern in the cluster and recalculate the new center of the cluster. Exit the loop.
5. If the pattern is not included in any cluster then create a new cluster with center in this pattern.

Algorithm 4. This algorithm is proposed in reference [4]. However, we have slightly modified the algorithm, in the original reference it is used a truncation for the Gaussian units and a hard limiting function for the output layer. We have applied the algorithm without these modifications of the normal RBF network.

The description of the algorithm is as follows. The Gaussian units are generated class by class k, so the process is repeated for each class. In a similar way to algorithm 2 the Gaussian units are generated in stages h. A stage is characterized by its majority criterion ρ_h, a majority criterion of 60% implies that the cluster of the Gaussian unit must have at least 60% of the patterns belonging to its class, this percentage of patterns is called $PC_j^r(k)$. The method will have a maximum of six stages; we begin with a majority criterion h of 50% and end with 100%, by increasing 10% in each stage $\Delta\rho$. The

Procedure 3. Algorithm 4

0. Initialize constants:
   (a) $delta_{max} = 10\rho$,
   (b) $\Delta\theta =$ some constant (10% in the reference)
   (c) $\delta_0 =$ some constant (0 or $0.1 \cdot \rho$),
   (d) $\Delta\delta = (\delta_{max} - \delta_0)/s$, ($s = 25$ in the reference).

1. Initialize class counter: $k = 0$.
2. Increment class counter: $k = k + 1$. If $k > K$, stop. Else, initialize cumulative Gaussian counters: $S_k = 0$ (empty set), $q = 0$.
3. Initialize stage counter: $h = 0$.
4. Increment stage counter: $h = h+1$. Increase majority criterion: If $h > 1$, $\phi_h = \phi_{h-1} + \Delta\phi$, otherwise $\phi_h = 50\%$. If $\phi_h > 100\%$, go to (2) to mask the next class.
5. Select Gaussian units for the hth stage: $j = 0$, $R = I$.
   (a) Set $j = j + 1, r = 1, r = 0$.
   (b) Select an input pattern vector x_i of class k at random from R, the remaining training set.
   (c) Search for all pattern vectors in R within a δ_r radius of x_i. Let this set of vectors be V_j^r.
      i. If $PC_j^r(k) < \phi_h$ and $r > 1$, set $r = r - 1$, go to (e).
      ii. If $PC_j^r(k) > \phi_h$ and $r > 1$, go to (d) to expand the neighborhood.
      iii. If $PC_j^r(k) < \phi_h$ and $r = 1$, go to (h).
      iv. If $PC_j^r(k) > \phi_h$ and $r = 1$, go to (d) to expand the neighborhood.
   (d) Set $r = r + 1$, $\delta_r = \delta_{r-1} + \Delta\delta$. If $\delta_r > \delta_{max}$, set $r = r - 1$, go to (e). Else, go to (c).
   (e) Remove class k patterns of V_j^r from R. If $N_j^r < \beta$, go to (g).
   (f) Set $q = q + 1$. Compute the center C_q^k and width w_q^k of the q-th Gaussian for class k. Add qth Gaussian to the set S_k. C_q^k = centroid of class k patterns in the set V_j^r w_q^k = standard deviation of the distances from the centroid C_q^k of the class k patterns in V_j^r.
   (g) If R is not empty of class k patterns, go to (a), else go to (6).
   (h) Remove class k patterns of the V_j^r from R. If R is not empty of class k patterns, go to (a), else go to (6).
6. From the set S_k, eliminate similar Gaussian units (those with very close centers and widths). Let Q_k be the number of Gaussian units after this elimination.
7. Calculate the output weights of the net for class k.
8. Compute TSE_h and TRE_h for class k. If $h = 1$, go to (4). Else:
   (a) If $TSE_h < TSE_{h-1}$, go to (4).
   (b) If $TSE_h > TSE_{h-1}$ and $TRE_h > TRE_{h-1}$, go to (4).
   (c) Otherwise, overfitting has occurred. Use the mask generated in the previous stage as class k mask. Go to (2) to mask next class.

Gaussian units for a given class k at any stage h are randomly selected in the following way. Randomly pick a pattern vector x_i of class k from the training set I and expand the radius of the cluster δ_r until the percentage of patterns belonging to the class k, $PC_j^r(k)$, falls below the majority criterion, then the patterns of class k are used to define the Gaussian unit (the mean C_q^k is the center and the standard deviation w_q^k is the width) and are removed from the training set, forming what it is called the remaining training set R. When the number of patterns in the cluster N_j^r is below than a parameter,

β, no Gaussian unit is created. To define the next gaussian another pattern x_i of class k is randomly picked from the remaining training set R and the process repeated. The successive stage process is repeated until the cross-validation error increases.

The algorithm can be summarized in the steps described in procedure 3. Where, ϕ is the maximum of the class standard deviations that are the standard deviations of the distances from the centroid of the patterns of each class, K is the number of classes, TRE_h is the training set error at stage h and TSE_h is the cross-validation set error.

3 Experimental Results

We have applied the training algorithms to nine different classification problems from the UCI repository of machine learning databases.

They are Balance Scale, Cylinders Bands, Liver Disorders, Credit Approval, Glass Identification, Heart Disease, The Monk's Problems and Voting Congresional Records. The complete data and a full description can be found in the repository http://www.ics.uci.edu/$\sim$ mlearn/MLRepository.html) [7].

3.1 Results

The first step was to determine the appropriate parameters of the algorithms by trial and error and cross-validation. We have used an extensive trial procedure and the final value of the parameters we have used in the experiments is in Table 1.

After that, with the final parameters we trained ten networks with different partition of data in training, cross-validation and test set, also with different random initialization of parameters. With this procedure we can obtain a mean performance in the database (the mean of the ten networks) and an error by standard error theory.

These results are in Table 2, 3, 4, 5 and 6. We have included for each database and training algorithm the mean percentage of correct classification with its error (column Perc.) and the number of gaussian transfer units under the column $Nunit$. In the case of unsupervised training algorithms number 2, 3 and 4 (Tables 5 and 6) the number in the column $Nunit$ is a value and an error. The reason is that the final number of Gaussian transfer units changes from one trial to another and we have included the mean value of the number of Gaussian units and the standard desviation as the error.

3.2 Interpretation of Results

Comparing the results of the same algorithm trained by gradient descent in the case of *Batch training* and *Online training*, we can see that the differences in performance are not significant. The fundamental difference between both training procedures is in the number of iterations and the value of the learning step. For example, 8000 iterations, η=0.001 in *EXP Batch* for Bala and 6000 iterations, η=0.005 in *EXP Online*. The final conclusion is that *online training* is more appropriate that *Batch training* for gradient descent of RBF.

Comparing *EXP*, *LIN* and *COS* generator functions, we can see that the general performance is quite similar except in the case mok1 where the performance of *EXP* is

Table 1. Optimal parameters of the different algorithms determined by trial and error

		Database				
Method	**Params.**	**bala**	**band**	**bupa**	**cred**	**glas**
EXP Batch	**Clusters**	45	110	35	40	125
	σ	0.6	1.2	0.6	1.8	0.4
EXP Online	**Clusters**	60	40	40	30	110
	σ	0.6	1	0.4	2	0.4
LIN Batch	**Clusters**	45	30	10	10	35
	γ	0.4	0.1	0.4	0.9	0.1
LIN Online	**Clusters**	50	35	15	10	30
	γ	0.6	0.3	0.8	0.1	0.2
COS Batch	**Clusters**	25	120	15	10	105
	a_j_ini	0.5	1.1	0.5	0.2	0.8
COS Online	**Clusters**	40	125	40	25	15
	a_j_ini	0.5	0.5	1.1	1.1	0.8
UC Alg.1	**Clusters**	30	60	10	20	100
	P	4	2	3	2	1
UC Alg.2	β	5	3	5	3	5
	α	0.8	0.65	0.8	0.8	0.8
UC Alg.3	β	5	7	5	6	5
	α	1.7	1.3	1.3	1.7	1.2
UC Alg.4	β	5	3	3	3	3

		Database			
Method	**Params.**	**hear**	**mok1**	**mok2**	**vote**
EXP Batch	**Clusters**	155	60	80	35
	σ	1.8	0.8	0.6	2
EXP Online	**Clusters**	20	30	45	5
	σ	2	0.8	0.6	1.8
LIN Batch	**Clusters**	15	15	25	25
	γ	0.3	0.2	0.5	0.1
LIN Online	**Clusters**	10	15	50	10
	γ	0.1	0.1	0.2	0.3
COS Batch	**Clusters**	25	100	125	20
	a_j_ini	0.2	0.2	0.2	0.5
COS Online	**Clusters**	15	145	45	10
	a_j_ini	0.5	1.1	0.2	0.2
UC Alg.1	**Clusters**	100	90	90	40
	P	1	2	2	4
UC Alg.2	β	3	3	8	3
	α	0.8	0.8	0.8	0.65
UC Alg.3	β	5	5	5	5
	α	1.7	1.7	1.4	1.7
UC Alg.4	β	3	3	3	5

clearly better. In other aspect, *EXP* and *LIN* functions need a higher number of trials for the process of trial and error to design the network, because cosine generator functions adapt all parameters. But in contrast, the number of iterations needed to converge

Table 2. Performance of Gradient Descent with Exponential Generator Functions

	Training Algorithm			
	Exp Batch		Exp Online	
Database	**Percentage**	$Nunit$	**Percentage**	$Nunit$
bala	90.2±0.5	45	90.2±0.5	60
band	74.1±1.1	110	74.0±1.1	40
bupa	69.8±1.1	35	70.1±1.1	40
cred	86.1±0.7	40	86.0±0.8	30
glas	92.9±0.7	125	93.0±0.6	110
hear	82.0±1.0	155	82.0±1.0	20
mok1	94.7±1.0	60	98.5±0.5	30
mok2	92.1±0.7	80	91.3±0.7	45
vote	95.6±0.4	35	95.4±0.5	5

Table 3. Performance of Gradient Descent with Linear Generator Functions

	Training Algorithm			
	Lineal Batch		Lineal Online	
Database	**Percentage**	$Nunit$	**Percentage**	$Nunit$
bala	90.1±0.5	45	90.6±0.5	50
band	74.5±1.1	30	73.4±1.0	35
bupa	71.2±0.9	10	69.7±1.3	15
cred	86.2±0.7	10	85.8±0.8	10
glas	91.4±0.8	35	92.4±0.7	30
hear	82.1±1.1	15	81.8±1.1	10
mok1	93.2±0.7	15	94.5±0.7	15
mok2	82.8±1.2	25	89.6±1.2	50
vote	95.6±0.4	25	95.6±0.4	10

Table 4. Performance of Gradient Descent with Cosine Generator Functions

	Training Algorithm			
	Cosine Batch		Cosine Online	
Database	**Percentage**	$Nunit$	**Percentage**	$Nunit$
bala	89.9±0.5	25	90.0±0.7	40
band	75.0±1.1	120	74.9±1.1	125
bupa	69.9±1.1	15	70.2±1.1	40
cred	86.1±0.8	10	86.1±0.8	25
glas	93.5±0.8	105	92.6±0.9	15
hear	82.1±1.0	25	81.9±1.1	15
mok1	89.8±0.8	100	90.2±1.0	145
mok2	87.9±0.8	125	86.6±1.1	45
vote	95.6±0.4	20	95.4±0.4	10

by *COS* functions is usually larger (for example: *EXP*, band= 10000 iterations; *LIN*, band= 15000; *COS*, band= 75000), so globally speaking the computational cost can be considered similar.

Table 5. Performance of Unsupervised Algorithms 1 and 2

	Training Algorithm			
	UC Alg. 1		UC Alg. 2	
Database	**Percentage**	$Nunit$	**Percentage**	$Nunit$
bala	88.5±0.8	30	87.6±0.9	88.5±1.6
band	74.0±1.5	60	67±2	18.7±1.0
bupa	59.1±1.7	10	57.6±1.9	10.3±1.5
cred	87.3±0.7	20	87.5±0.6	95±14
glas	89.6±1.9	100	79±2	30±2
hear	80.8±1.5	100	80.2±1.5	26±4
mok1	76.9±1.3	90	72±2	93±8
mok2	71.0±1.5	90	66.4±1.7	26±4
vote	95.1±0.6	40	93.6±0.9	53±5

Table 6. Performance of Unsupervised Algorithms 3 and 4

	Training Algorithm			
	UC Alg. 3		UC Alg. 4	
Database	**Percentage**	$Nunit$	**Percentage**	$Nunit$
bala	88.0±0.9	94.7±0.5	87.4±0.9	45±7
band	67±4	97.2±0.3	65.8±1.4	4.5±1.3
bupa	60±4	106.2±0.3	47±3	11±5
cred	87.9±0.6	161.10±0.17	86.4±0.9	32±4
glas	82.8±1.5	59.9±0.7	81.2±1.8	22±2
hear	72±4	71.8±0.6	78±3	10±2
mok1	68±3	97.4±0.6	64±2	23±6
mok2	66.5±0.8	143±0	71.6±1.5	20±2
vote	94.1±0.8	120.30±0.15	76±5	5.0±1.1

Table 7. Performance of Multilayer Feedforward with Backpropagation

Database	**N. Hidden**	**Percentage**
bala	20	87.6±0.6
Bands	23	72.4±1.0
bupa	11	58.3±0.6
cred	15	85.6±0.5
glas	3	78.5±0.9
hear	2	82.0±0.9
mok1	6	74.3±1.1
mok2	20	65.9±0.5
vote	1	95.0±0.4

In the original reference *LIN* and *COS* transfer functions were propossed as an improvement to the traditional *EXP* transfer function. We have not observed any improvement in our results.

Comparing unsupervised training algorithms among them, it seems clear that the classical algorithm 1, k-means clustering shows the better performance.

Finally, comparing unsupervised training with gradient descent we can see that the best alternative (under the performance point of view) is supervised training by gradient descent, it achieves a better performance in 6 of 9 databases.

In order to perform a further comparison, we have included the results of Multilayer Feedforward with Backpropagaion in Table 7. We can see that the results of RBF are better. This is the case in all databases except cred, hear and vote where the performance of both networks is similar.

4 Conclusions

In this paper we have presented a comparison of unsupervised and fully supervised training algorithms for RBF networks. The algorithms are compared using nine databases. Our results show that the fully supervised training by gradient descent may be the best alternative under the point of view of performance. The results of RBF are also compared with Multilayer Feedforward with Backpropagation.

In the case of fully supervised training algorithms we have performed experiments with three different transfer functions in the hidden units and the performance is similar. We have not observed an improvement in performance with *LIN* and *COS* functions as pointed out in the bibliography.

Furthermore under the point of view of training speed the alternative of *Online Training* is better than *Batch Training*.

Finally, we have included the performance on the same datasets of the network Multilayer Feedforward with Backpropagation and it seems that the performance of RBF trained by Gradient Descent is in general better.

Acknowledgments

This research was supported by project *P1 ·1B2004-03* of Universitat Jaume I - Bancaja in Castellón de la Plana, Spain.

References

1. J. Moody, and C.J. Darken, “Fast Learning in Networks of Locally-Tuned Procesing Units.” *Neural Computation*, vol.1, pp 281-294, 1989.
2. A. Roy, S. Govil et alt, “A Neural-Network Learning Theory and Polynomial Time RBF Algorithm.” *IEEE Trans. on Neural Networks*, vol.8, no. 6, pp.1301-1313, 1997.
3. Y. Hwang and S. Bang, “An Efficient Method to Construct a Radial Basis Function Neural Network Classifier.” *Neural Network*, Vol.10 no. 8, pp.1495-1503, 1997
4. A. Roy, S. Govil et alt, “An Algorithm to Generate Radial Basis Function (RBF)-Like Nets for Classification Problems.”, *Neural Networks*, vol.8, no. 2, pp.179-201, 1995.
5. N. Krayiannis, “Reformulated Radial Basis Neural Networks Trained by Gradient Descent.” *IEEE Trans. on Neural Networks*. vol.10, no. 3, pp.657-671, 1999.

6. N. Krayiannis and M. Randolph-Gips, "On the Construction and Training of Reformulated Radial Basis Functions." *IEEE Trans. Neural Networks*. vol.14, no. 4, pp.835-846, 2003.
7. D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz, "UCI Repository of machine learning databases.", http://www.ics.uci.edu/∼mlearn/MLRepository.html, University of California, Irvine, Dept. of Information and Computer Sciences, 1998.