Hybrid Learning of RBF Networks

Roman Neruda* and Petra Kudová

Institute of Computer Science, Academy of Sciences of the Czech Republic, P.O. Box 5,
18207 Prague, Czech Republic
roman@cs.cas.cz

Abstract. Three different learning methods for RBF networks and their combinations are presented. Standard gradient learning, three-step algoritm with unsupervised part, and evolutionary algorithm are introduced. Their perfromance is compared on two benchmark problems: Two spirals and Iris plants. The results show that three-step learning is usually the fastest, while gradient learning achieves better precission. The combination of these two approaches gives best results.

1 Introduction

By an *RBF unit* we mean a neuron with multiple real inputs $\vec{x} = (x_1, \ldots, x_n)$ and one output y. Each unit is determined by an n-dimensional vector $\vec{c}$ which is called *center*. It can have an additional parameter b usually called *width*.

The output y is computed as:

$$y = \varphi(\xi); \quad \xi = \frac{\| \vec{x} - \vec{c} \|}{b} \tag{1}$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ is a suitable activation function, typically Gaussian $\varphi(z) = e^{-z^2}$.

For evaluating $\frac{||\vec{x}-\vec{c}||}{d}$, the Euclidean norm is usually used. In this paper we consider a general weighted norm instead of the Euclidean norm. A weighted norm is determined by a $n \times n$ matrix $\mathbf{C}$ and is defined as

$$\| \vec{x} \|_C^2 = (\mathbf{C}\vec{x})^T(\mathbf{C}\vec{x}) = \vec{x}^T\mathbf{C}^T\mathbf{C}\vec{x}. \tag{2}$$

It can be seen that the Euclidean norm is a special case of a weighted norm determined by an identity matrix. In further text we will use the symbol $\mathbf{\Sigma^{-1}}$ instead of $\mathbf{C}^T\mathbf{C}$.

In order to use a weighted norm each RBF unit has another additional parameter matrix $\mathbf{C}$.

An *RBF network* is a standard 3-layer feedforward network with the first layer consisting of n input units, a hidden layer consisting of h RBF units and an output layer of m linear units. Thus, the network computes the following function $\vec{f} : \mathbb{R}^n \to \mathbb{R}^m$:

$$f_s(\vec{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\| \vec{x} - \vec{c}_j \|_{C_j}}{b_j}\right), \tag{3}$$

* This work has been partially supported by GACR under grants 201/00/1489 and 201/02/0428.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2331, pp. 594–603, 2002.

where $w_{ji} \in \mathbb{R}$ and f_s is the output of the s-th RBF unit.

Denote $T = \{(\vec{x}(t), \vec{d}(t); t = 1, \ldots, k\}$ a *training set* — a set of examples of network inputs $\vec{x}(t) \in \mathbb{R}^n$ and desired outputs $\vec{d}(t) \in \mathbb{R}^m$. For every training example we can compute the actual network output $\vec{f}(\vec{x}(t))$ and error $e_j(t)$ of each of the output units:

$$e_j(t) = d_j(t) - f_j(t).$$

The instantaneous error $\mathcal{E}(t)$ of the whole network is then:

$$\mathcal{E}(t) = \frac{1}{2} \sum_{j=1}^{p} e_j^2(t). \tag{4}$$

The goal of learning an RBF net is to minimize an error function

$$E = \sum_{t=1}^{k} \mathcal{E}(t). \tag{5}$$

1.1 Three step learning

The gradient learning described in previous section unifies all parameters by treating them in the same way. Now we introduce a learning method taking advantage of the well defined meaning of RBF network parameters (cf. [1], [2]).

There are three categories of RBF network parameters, so we can divide the learning into three consequent steps and customize the method of each step for the appropriate parameter.

The first step consists of determining the hidden unit centers. The positions of centers should reflect the density of data points and thus various clustering or vector quantization techniques can be used. Using a genetic algorithm during the first step will be discussed in 1.2.

The second phase sets the additional hidden unit parameters if there are any. There can be a parameter called width or a weighted norm matrix. These parameters determine the size and the shape of the area controlled by the unit. Suitable parameter values can be found by gradient minimization of function

$$E(b_1, \cdots, b_h; \mathbf{\Sigma_1^{-1}}, \cdots, \mathbf{\Sigma_h^{-1}}) = \tfrac{1}{2} \sum_{r=1}^{h} \left[\sum_{s=1}^{h} \varphi\left(\xi_{sr}\right) \xi_{sr}^2 - P \right]^2 \tag{6}$$

$$\xi_{sr} = \frac{\parallel c_s - c_r \parallel_{C_r}}{b_r}$$

where P is the overlap parameter controlling the overlap between areas of importance belonging to particular units.

In case of units with widths we can get around the minimization using simple heuristics. The often used one called the q-neighbours rule simply set the width proportionally to the average distance of q nearest neighbouring units.

The third step is a usual supervised learning known from multilayer perceptron networks reduced to a linear regression task. The only parameters to be set are the

weights between the hidden and the output layer which represent the coefficients of linear combinations of RBF units outputs. Our goal is to minimize the overall error function:

$$E = \frac{1}{2}\sum_{t=1}^{k}\sum_{s=1}^{m}(d_s^{(t)} - f_s^{(t)})^2 \quad . \tag{7}$$

It can be achieved using gradient minimization or assuming the partial derivative $\frac{\partial E}{\partial w_{ij}}$ equal to zero and finding the solution in terms of linear optimalization using any of various linear least squares methods.

$$\sum_{t=1}^{k}(d_r^{(t)} y_q(\vec{x}^{(t)})) - \sum_{j=1}^{h} w_{jr} \sum_{t=1}^{k}\left(y_j(\vec{x}^{(t)}) y_q(\vec{x}^{(t)})\right) = 0 \quad , \tag{8}$$

where $q = 1, \ldots, h$ and $r = 1, \ldots, m$.

It is true, however, that the success of this learning step depends on the previous steps.

1.2 Evolutionary learning

The third learning method is based on using a genetic algorithm. It is a stochastic optimization method inspired by evolution, using principals as selection, crossover and mutation.

A genetic algorithm works with a *population* of *individuals*. An individual (see fig. 5) represents some feasible values for all parameters of an RBF net being learned. Each individual is associated with the value of the error function of a corresponding network.

Starting with a population of random individuals new populations are produced using operators of *selection*, *mutation* and *crossover*. The *selection* guarantees that the possibility of being chosen to the new population is the higher the smaller is the error function of the corresponding network. The *crossover* compose a pair of new individuals combining parts of two old individuals. The *mutation* brings some random changes into the population. We iterate until population contains an individual with an error small enough.

Genetic algorithm can be combined with previous methods. Specifically, the determination of centers in the three-step method can be done by means of genetic algorithm. Than an individual codes only values of centers and its error is computed as

$$E_{vq} = \frac{1}{k}\sum_{t=1}^{k} \parallel \vec{x}^t - \vec{c}_c{}^2 \parallel, \quad c = \mathrm{argmin}_{i=1,..h}\{\parallel \vec{x}^t - \vec{c}_i \parallel\}, \tag{9}$$

where $\vec{x}^t$ is training sample a $\vec{c}_i$ is the center of ith unit.

We implemented also canonical version of genetic learning described in [3].

2 Experiments

In the following sections results from our experiments will be presented. The first is a classification task, called *Two Spirals*. We will demonstrate an advantage of using a weighted norms. The second task, also a clasification – the known *Iris Plants*, compare all three methods described in previous sections and shows the advantage of combining two of them, specifically the three step method and the gradient learning.

All experiments were run on the Linux cluster. Each computation was run on an individual node with a Celeron 533 MHz processor and 384 MB memory.

2.1 Two spirals

The task of the first experiment, *Two Spirals*, is to discriminate between two sets of training points which lie on two distinct spirals in the 2D plane. The training set contains 372 training samples, each 2 input values (2D coordinates) and 1 output value (classification – either value 0.0 or value 1.0).

Considering the character of the training data we expect that a rather high number of RBF units will be needed. We used a network with 150 RBF units and both the Euclidean norm and a weighted norm. This network was trained using the gradient learning and the three step learning. The genetic algorithm isn't suitable because of the higher number of RBF units.

Gradient learning Knowing that a result of the gradient learning is dependent on the initial setup of parameters, the gradient learning was run five times using the Euclidean norm and five times using weighted norms and we consider the average, the worst and the best computation.

All computations were stopped after 5 000 iterations, the average time of 100 iteration was 361 seconds for an RBF net with weighted norms and 115 seconds for an RBF net with the Euclidean norms. In Figure 1 you can see the fall of the error function for the average computation using the Euclidean norm and for the average computation using weighted norms. The average error after 5000 iterations was 0.0167 for Euclidean norm, and 0.0088 for the weighted norm. Table 1 compares the time and the number of iterations needed for the fall of the error function under a given ε with an RBF net using the Euclidean norm and an RBF net using a weighted norm.

However a computation using weighted norms is slower than using the Euclidean norms, fewer iterations are needed to reach a given error and in the end a better solution is obtained.

Three step learning The three step learning was the second method applied on *Two Spirals*. Since the training samples are distributed on two spirals, we used several vector quantization methods to distribute the centers of RBF units. The Lloyd algorithm and the k-means clustering are known vector quantization methods. The genetic algorithm is our application of a common genetic algorithm to vector quantization.

The resulting value of the error function is comparable for all methods (see Table 2), but in case of the genetic algorithm there is the least number of unused units. However, the genetic algorithm has much higher time requirements.

The second step was realized by a gradient minimization of the error function (see section 1.1), 200 iterations (17 seconds for Euclidean norms, 90 seconds for weighted norms) were needed. For the determination of weights a linear least squares method was used (16 seconds for Euclidean norms, 90 seconds for weighted norms). The errors of the RBF nets learned by the three step learning are 1.101 for Euclidean norm, and 0.051 for the weighted norm.

We used two different methods to learn the *Two Spirals problem*. In both of them we saw the difference between the RBF net using Euclidean norms and the RBF net using weighted norms. In both of them the RBF net using weighted norm has a smaller error. We can interpret a use of a weighted norm as a transformation of a radial field of an RBF unit to an oval one. Then covering an input space by ovals is easier than using circles.

2.2 Iris

In the second experiment we used a well-known data set *Iris Plants*. It contains 3 classes of 50 instances each, where each class refers to a type of an iris plant. One class is linearly separable form the others, the other are not linearly separable from each other.

We used a net with three output neurons, one neuron for each class. The class number is then coded by three binary values, value 1 on the position corresponding with the number of class and zeros on the others. So each training sample consists of 4 input values describing some features of a plant and 3 output values coding its class.

We split *The Iris Plants* data set into two parts. The first containing 120 instances (40 per class) is used as a training set, the second containing other 30 instances (chosen randomly) is used for testing.

We applied all three methods (the gradient learning,the three step method and the genetic learning) on an RBF net with 3, 6 and 9 hidden units, all with weighted norms.

Gradient learning The gradient algorithm was run five times and the average, the minimum and the maximum computation was picked up. Figure 3 compares the fall of the average gradient algorithm error function. The number of iterations needed to reach a given error is shown in Table 3. Table 4 shows the error of the learned RBF net on the training set and the testing set with the number of misclassified samples.

Three step learning The three step learning consisted of a vector quantization using both the Lloyd algorithm and the genetic algorithm, the gradient minimization in the second step and the linear least squares.

Figures 3 and 4 show the fall of the vector quantization error function for the Lloyd algorithm and the genetic learning. 100 iterations of the genetic algorithm needs 3 seconds, 1000 iterations of the Lloyd algorithm need 2 seconds. Table 5 you see the resulting error (the average, the minimum, the maximum of five computations). 10 iterations of the Lloyd algorithm or 2000 iterations of the genetic algorithm were needed. The

genetic algorithm is better than the Lloyd algorithm, which is highly dependent on its random initialization. However, the time requirements of the genetic algorithm are much higher.

In the second step a gradient minimization was used (1000 iterations, 1s) and in the third step the linear least squares (1s). In Table 6 you see the resulting errors and the numbers of misclassified samples.

Because of the resulting error of the three step learning is much worse than the one of the gradient learning, we decided to add a fourth step. Specifically we used the RBF net learned by the three step learning as the initialization for the gradient learning. Then the fourth step consists of some iterations of gradient learning, we practiced 5 000 iterations. The fall of the error function is shown on Figure 4, the number of iterations and the time needed to reach the given error is in Table 7. The review of the resulting errors is in Table 8.

Although the results of the three step learning were not the best, its time requirements are very low and so it can be used successfully as initialization of the gradient learning.

Genetic learning We ran the genetic algorithm five times and consider the average, the minimum and the maximum computation. All computations worked with a population of 50 individuals, an elite of 2 individuals and the mutation rate 0.2, the crossover rate 0.7. The average time of 100 iterations was 32.8 s using 3 units, 114 s using 6 units and 149 s using 9 units. In Figure 5 see the fall of the error function, Table 9 shows the number of iterations and time needed to reach a given error. The review of resulting errors is in Table 10.

The genetic algorithm is a little bit worse than previous methods. Its great disadvantage are its time requirements.

All three methods described in section 1 were demonstrated on the *Iris Plants* task. The genetic learning ended with a higher error and the highest time requirements. The gradient learning converged to the lowest error. The three step method has the very lowest time requirements. We showed that the best way is to apply the three step learning followed by the gradient learning.

References

1. K. Hlaváčková and R. Neruda. Radial basis function networks. *Neural Network World*, 3(1):93–101, 1993.
2. J. Moody and C. Darken. Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo, CA, 1989. Morgan Kaufmann.
3. R. Neruda. *Functional Equivalenece and Gentic Learning of RBF Networks*. PhD thesis, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 1998.
4. Kudová P. Neuronové sítě typu RBF pro analýzu dat. Master's thesis, Charles University, Prague, Faculty of Mathematics and Physics, 2001.

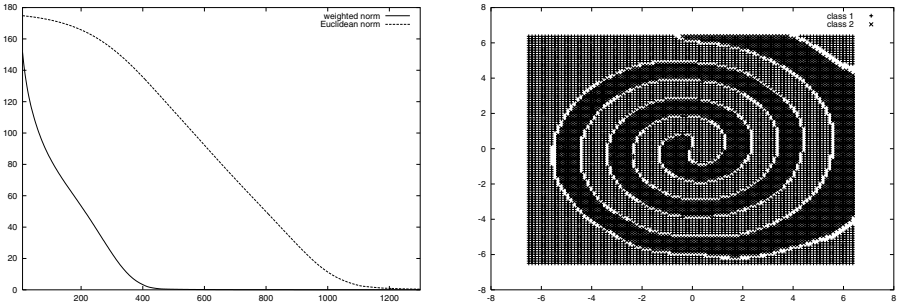

Fig. 1. Two spirals: a) Te gradient learning error function. b) The network output partitioning the intput space.

Table 1. Two spirals: The average number of iterations and time to reach a given ε.

	Euclidean norm		Weighted norm	
ε	iterations	time	iterations	time
10	1011	19 min 23 s	353	21 min 17 s
1	1192	22 min 51 s	441	26 min 35 s
0.5	1272	24 min 24 s	470	28 min 20 s
0.1	1929	37 min 0 s	689	41 min 32 s
0.01	–	–	3829	3 hour 50 min 49 s

Table 2. Two spirals: Vector quantization, 1st step.

	error	number of passes through the trainset	time
Lloyd alg.	0.1296	8	1 s
K-means clustering	0.1066	50	9 s
K-means cl. with local memory	0.0940	200	1 min 6 s
Genetic algorithm	0.1183	$50 \times 2\,500$	4hours 31 min

Table 3. Iris: Average number of iterations and time to reach a given ε.

	3 units		6 units		9 units	
ε	iterations	time	iterations	time	iterations	time
100	1	$<$ 1s	1	$<$ 1s	1	$<$ 1s
50	5	1s	2	1s	176	38s
10	1832	1 min 49s	141	14s	427	1 min 33s
5	1833	1 min 49s	695	1 min 9s	445	1 min 37s
3	—	—	—	—	1380	5 min 3s

Table 4. Iris: The error divided by the number of samples and the number of misclassified samples.

	Error on trainset			Error on test set		
	average	minimum	maximum	average	minimum	maximum
3 units	0.029 (0)	0.026 (0)	0.034 (1)	0.092 (2)	0.089 (2)	0.099 (2)
6 units	0.037 (0)	0.017 (0)	0.065 (4)	0.100 (2)	0.087 (2)	0.123 (2)
9 units	0.020 (0)	0.010 (0)	0.025 (0)	0.106 (2)	0.090 (2)	0.123 (2)

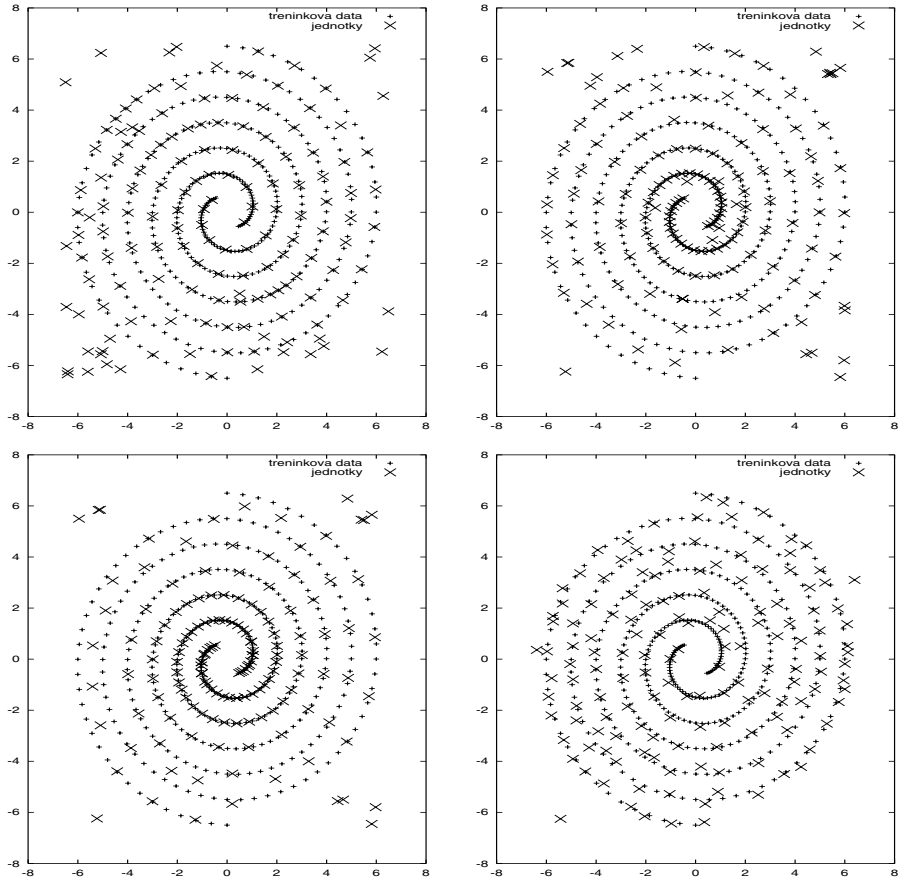

Fig. 2. Lloyd algorithm. K-means clustering. K-means clustering with local memory. Genetic algortihm.

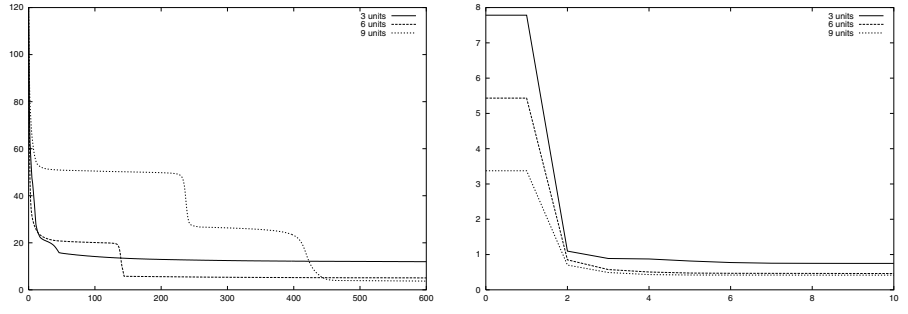

Fig. 3. Iris: a) The gradient learning error function using an RBF net with 3, 6 and 9 units. b) The VQ learning error function – 1st step (3, 6 and 9 units)

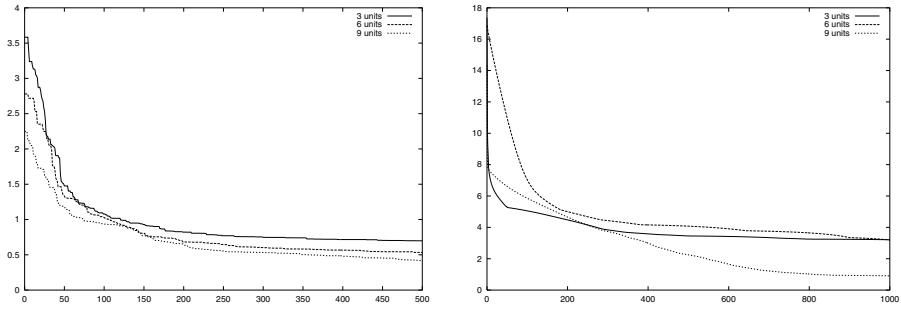

Fig. 4. Iris: a) The genetic learning error function – 1st step (3, 6 and 9 units). b) The 4th step gradient learning error.

Table 5. Iris: The error of 1st step using the Lloyd algorithm and the Genetic algorithm

	3 units			6 units			9 units		
	average	min	max	average	min	max	average	min	max
Lloyd alg.	0.748	0.497	1.000	0.460	0.355	0.498	0.416	0.340	0.497
Genetic alg.	0.650	0.499	0.977	0.449	0.326	0.666	0.343	0.243	0.425

Table 6. Iris: The three step learning error divided by the number of samples.

	Error on trainset	Error on testset
3 units	0.14 (13)	0.20 (6)
6 units	0.14 (18)	0.18 (6)
9 units	0.12 (14)	0.17 (5)

Table 7. Iris: The gradient learning (after the three steps) – the number of iterations needed to reach a given ε

	3 units		6 units		9 units	
ε	iterations	time	iterations	time	iterations	time
10	2	< 1	61	6s	1	< 1
5	111	6s	201	20s	171	37s
1	–	–	1687	2 min 48s	813	2 min 58s
0.5	–	–	–	—	3537	12 min 58s

Table 8. Iris: 4th step gradient learning error divided by number of samples

	Error on trainset	Error on trainset
3 units	0.0244 (0))	0.092 (2)
6 units	0.0078 (0)	0.136 (2)
9 units	0.0018 (0)	0.109 (2)

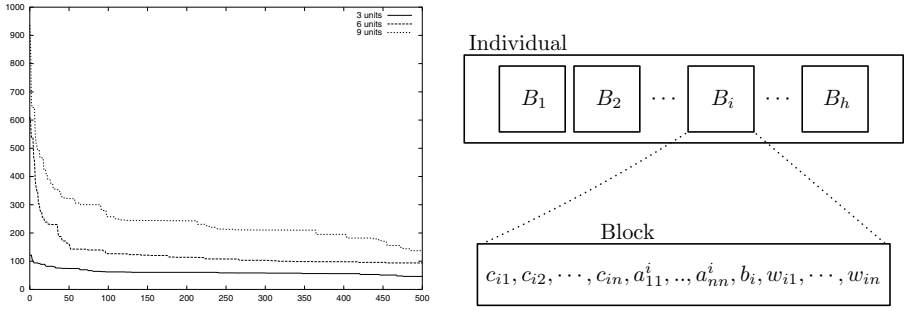

Fig. 5. a) The genetic learning error function using an RBF net with 3, 6 and 9 units.b) An individual representing an RBF net.

Table 9. The average number of iterations and time to reach a given ε.

	3 units		6 units		9 units	
ε	iterations	time	iterations	time	iterations	time
100	5	1s	317	6 min 4s	963	24 min 4s
50	468	2 min 29s	5443	1hour 46 min 14s	7515	3hours 7 min 52s
40	2325	12 min 24s	7835	2hours 30 min 10s	15990	6hours 39 min 45s
30	8859	47 min 14s	15925	5hours 5 min 13s	39819	16hours 35 min 28s
20	—	—	52753	16hours 51 min 5s	—	—

Table 10. The error divided by number of samples, number of misclassified samples.

	Error on trainset			Error on test set		
	average	minimum	maximum	average	minimum	maximum
3 units	0.249 (11)	0.134 (5)	0.361 (17)	0.225 (3)	0.182 (1)	0.292 (6)
6 units	0.161 (6)	0.089 (0)	0.284 (8)	0.272 (3)	0.165 (2)	0.429 (5)
9 units	0.242 (10)	0.205 (6)	0.335 (16)	0.399 (4)	0.223 (1)	0.662 (7)