Technical Report, October 2002

UNIVERSITY OF ESSEX

Department of Computer Science

TECHNICAL REPORT
CSM-412

CMAC and ANFIS for Nonlinear Modelling

Eric M. Rosales and Qiang Gan
emrosa@essex.ac.uk*, *jqgan@essex.ac.uk

Technical Report, October 2002

***Abstract*:** Manipulator control is one of the main research areas in robotics, demanding for models with fast convergence and reliable stability for modelling and control. In this paper, the mechanisms of CMAC and ANFIS models for input space partitioning are analysed and their performance is compared to each other in terms of their abilities to model complex non-linear processes, with performance indexes such as accuracy, convergence speed, and computational cost. Meanwhile, some guides on how to choose model parameters are presented. Also, an analysis of the CMAC with linear functional weights is presented as an improvement to the CMAC model.

1. Introduction

When modelling a function or controlling a manipulator, some critical issues that should be taken into account include (a) good approximation ability, (b) low computational cost, and (c) good fitness with existing methods in the subsequent applications. Various models for modelling and control have been proposed, such as local models CMAC (Cerebellar Model Articulation Control) by Albus [1-5], ANFIS (Adaptive-Network-based Fuzzy Inference System) by Jang [6], and ASMOD (Adaptive Spline Modelling) by Kavli [7]. Among the above three local models, ANFIS has the advantage of good applicability as it can be interpreted as local linearisation modelling and conventional linear techniques for state estimation and control are directly applicable, CMAC is reputed for its fast training speed, and ASMOD is good at achieving parsimonious structure for specific applications due to its algorithm for automatic input space partitioning. Miller et al. [8-10] and Pak-Cheung [11] have investigated on CMAC and made some useful improvements. Nelles [12] has proposed an algorithm called LOLIMOT (Local Linear Model Tree) for automatic input space partitioning in ANFIS type models. Gan and Harris [13-14] have extended the ASMOD algorithm to local linear modelling using B-splines, which has been successfully applied to nonlinear state estimation.

This paper analyses the mechanisms of CMAC and ANFIS in input space partitioning, compares them in terms of approximation accuracy, training speed, and memory requirement, aiming at combining the advantages of CMAC and ANFIS, and then proposes a CMAC model with linear functional (LF) weights. The CMAC and ANFIS models are briefly described and analysed in section 2. Experimental results are given and compared in section 3, with remarks on the guidelines to choose the CMAC parameters and the possible improvement. Section 4 shows some results of using the CMAC with LF weights for nonlinear modelling, and the local approximation property of the CMAC with LF weights is discussed in section 5.

2. Model Description and Analysis

An ANFIS model can be described by the following equation:

$$y = \sum_{j=1}^{R} \frac{\mu_{A_j}(x_1, \cdots, x_n)}{\sum_{l}^{R} \mu_{A_l}(x_1, \cdots, x_n)} (w_{j0} + w_{j1}x_1 + \cdots w_{jn}x_n) \tag{1}$$

where y is the output, x_i, $i=1,\ldots,n$, are the inputs, $\mu_{A_j}(x_1 \cdots x_n)$ the membership functions, w_{ji} the adjustable weights, and R is the number of fuzzy rules. It can be described as a multilayered feedforward network as shown in Fig. 1, where layer L1 executes a fuzzification process, layer L2 executes the fuzzy AND of the antecedent part of the fuzzy rules, layer L3 normalises the membership functions, layer L4 executes the consequent part of the fuzzy rules, and the final layer computes the output by summing up the outputs of layer L4. Gaussian or bell-shaped functions are used as membership functions in ANFIS, whose parameters are usually determined by the gradient-descent algorithm. The parameters in the consequent part are determined by the least mean square (LMS) or Kalman filtering algorithm.

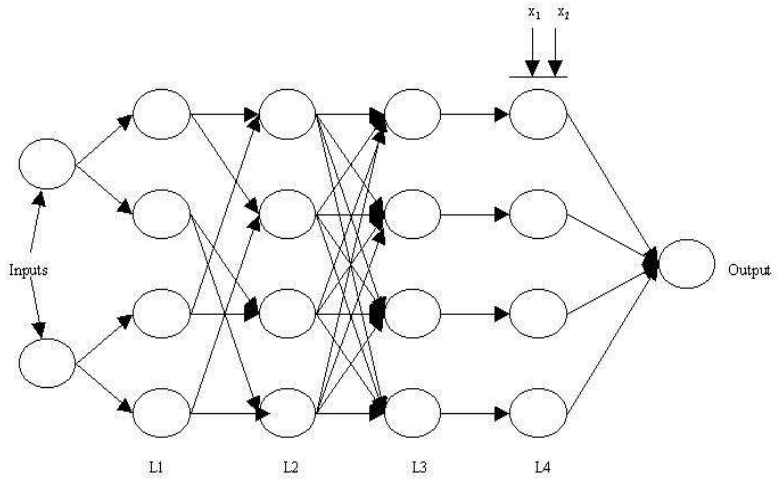

Figure 1 ANFIS architecture with two inputs, two membership functions per input, and one output

The equation for describing a CMAC model is as follows:

$$y = \sum_{j=1}^{C} \frac{f(\delta_j(x_1, \cdots, x_n))}{\sum_{l=1}^{C} f(\delta_l(x_1, \cdots, x_n))} w_{A_j} \tag{2}$$

where $f(\delta_j(x_1, \cdots, x_n))$ represents receptive field (RF) functions that play a similar role as the membership functions in ANFIS, $\delta_j(x_1, \cdots, x_n)$ maps the input vector **x** into a RF field address in the *j*th layer of RFs, w_{A_j} are the adjustable weights, A_j represents the address of an adjustable weight associated with the RF in the *j*th RF layer that is determined by a mapping process and a hashing function, and *C* is the number of RF layers, corresponding to the number of fuzzy rules in ANFIS. In the CMAC model, only one receptive field in each RF layer will be activated by the input, and only those weights associated with the activated RFs will be updated using the LMS-type algorithm. Usually for the same task the number of RF layers in CMAC is much smaller than the number of fuzzy rules in ANFIS. That is the main reason why the CMAC training converges very fast. Similarly, a CMAC model can also be depicted as a multilayered feedforward network. Details about how the ANFIS and CMAC models partition the input space and the inherent curse of dimensionality problem will be analysed in the following subsections.

2.1. Input Space Partitioning

In order to model complex nonlinear systems, both CMAC and ANFIS models carry out input space partitioning that splits the input space into many local regions on which simple local models (linear functions or even adjustable coefficients) are employed. ANFIS uses fuzzy membership functions for splitting each input dimension, as shown in Fig. 2, the input space is covered by overlapping membership functions that means several local regions can be activated simultaneously by a single input. For multidimensional space, the partitioning is defined by the fuzzy AND, such as tensor product, of univariate membership functions.

In CMAC the input space is split by receptive fields that are organised into multiple layers. The RFs in each layer do not overlap, but the RF layers are placed with offsets, as shown in Fig. 3 with 4 layers of triangle-shaped functions as RFs. Corresponding to an input, there will be only one RF in each layer activated. However, the activated RFs in all the layers have a similar overlapping effect in covering local regions of the input space, as in ANFIS.

As simple local models are adopted in both ANFIS and CMAC models, their approximation ability will depend on the resolution of the input space partitioning, which is determined by the number of membership functions in

ANFIS and the number of layers, i.e., the generalisation parameter C, and the size of offsets that depend on quantisation parameters q_i in CMAC.

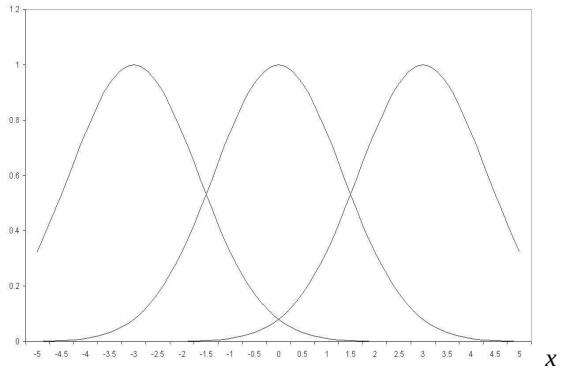

Figure 2 Input space partitioning in ANFIS

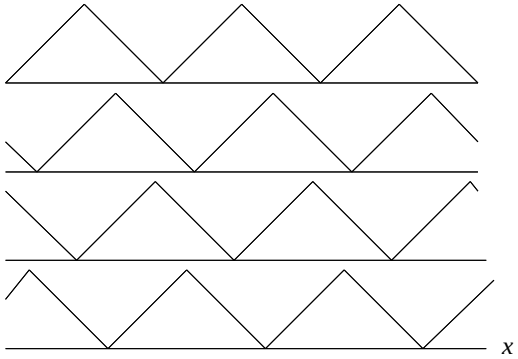

Figure 3 Input space partitioning in CMAC

2.2. The Curse of Dimensionality

In both ANFIS and CMAC models, the number of adjustable weights increases exponentially with the dimension of the input space. This is called the curse of dimensionality problem. The base of the exponent depends on the number of membership functions in ANFIS or the size of the offsets in CMAC. To overcome the curse of dimensionality problem, the ASMOD algorithm [7] and its modified version [13] automatically search the minimum number of required membership functions. The LOLIMOT [12] algorithm can do the similar search for ANFIS. The CMAC model uses a hashing function to combat this problem, in which the number of adjustable weights may be huge, but only limited number of weights will be activated and updated in correspondence to an input.

By comparing the different approaches used in ANFIS and CMAC for partitioning the input space and combating the curse of dimensionality problem, this paper aims at combining the advantages from both models. This will be discussed further after presenting some interesting experimental results.

3. Nonlinear Function Modelling

How good is the CMAC for modelling nonlinear systems? To answer this question, an experiment was set up to measure the performance of ANFIS and CMAC in modelling nonlinear functions. The training and testing data were generated using the following nonlinear function:

$$y = \frac{\sin(x_1)}{x_1} \cdot \frac{\sin(x_2)}{x_2} \tag{3}$$

A series of training data sets of 50x50 data points were generated with (x_1, x_2) randomly distributed and some noise added, each set was used in a different training epoch, which means that in each epoch some new training data were firstly presented and some training data used in the previous epochs were repeated. The testing data were generated with (x_1, x_2) uniformly distributed. A typical training data set is shown in Fig. 4, and the testing data is shown in Fig. 5, where $-10<x_1, x_2<10$.

Various CMAC parameter settings were tested, with generalisation parameter C=16, 32, or 64, and quantisation parameter q_i=0.05, 0.075, or 0.1. The choice of these parameter values are similar to those used in [8]. With 2-dimensional inputs, there are 27 combinations of quantisation and generalisation parameter choices. In the experiment various learning rates have also been examined. In total, 108 CMAC runs were carried out. In the experiment with ANFIS, the number of univariate membership functions varies from 5, 8, to 10. The training processes for both CMAC and ANFIS stop when either the preset maximum number of epochs or root mean square error (rmse) of test-set validation is satisfied. In the experiment, the maximum number of epochs was set to 80 and rmse threshold set to 0.003.

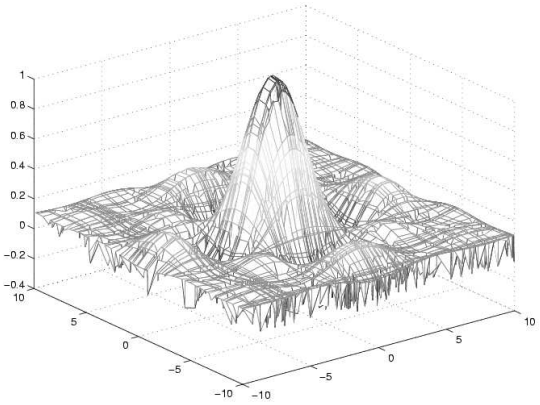

Figure 4 A typical training data set

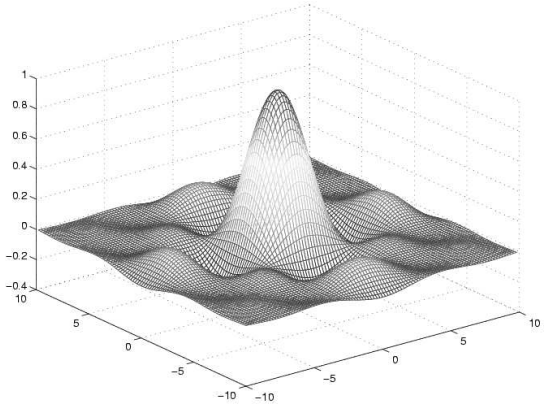

Figure 5 Testing data set

3.1 Approximation Ability

How good is the CMAC's approximation ability in comparison with ANFIS? Both ANFIS and CMAC, with appropriate parameter settings, have good approximation ability. Fig. 6 shows an approximation result of the CMAC on the testing data. Its approximation error is shown in Fig. 7. In a similar way, an approximation result of the ANFIS is shown in Fig. 8 and Fig. 9. With both CMAC and ANFIS models, the errors are not visually noticeable from the network output. The error figures show that both models have achieved very good approximation accuracy; CMAC performs a bit better than ANFIS though. This can be noticed by looking carefully at the error range of each model.

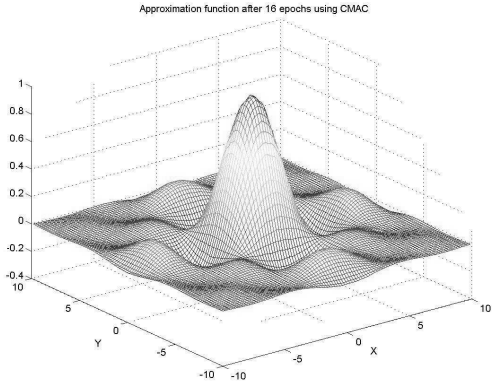

Figure 6 Approximation by CMAC

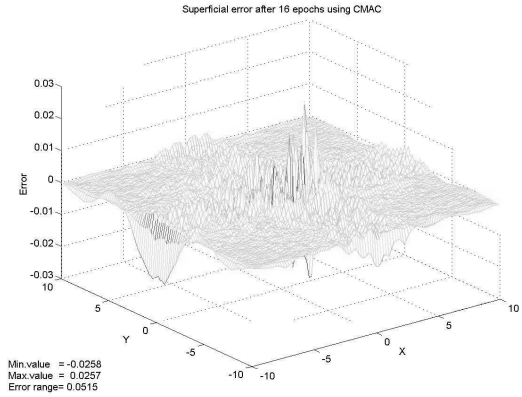

Figure 7 CMAC's error

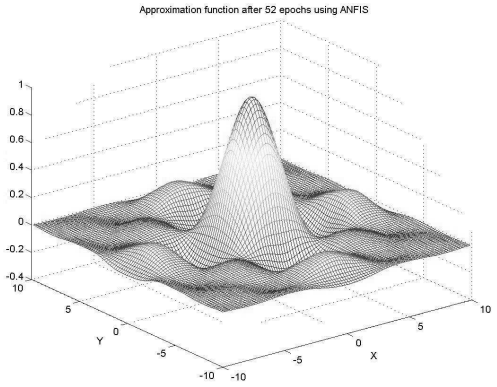

Figure 8 Approximation by ANFIS (10 membership functions)

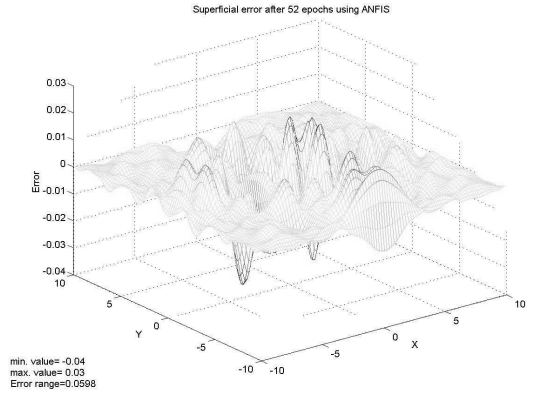

Figure 9 ANFIS's error

How does the CMAC's quantisation affect its approximation ability? An analysis of three different runs, where the generalisation level and one of the two quantisation values were fixed (C=16, q_1=0.075), shows that greater quantisation values lead to faster convergence, as shown in Fig 10. However, it is noted that medium quantisation values (0.075-0.075) produce a better approximation. The relationship between the approximation ability and the quantisation values is complicated. Smaller quantisation values correspond to finer input space partitioning. However, experimental results show that too small or too large quantisation values do not produce the best result.

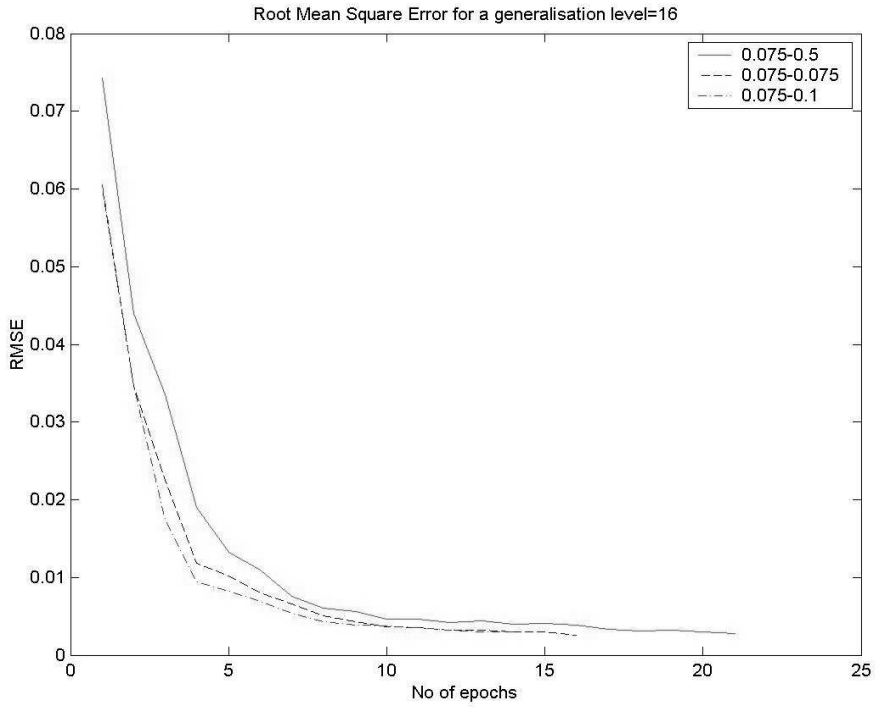

Figure 10 RMSE behaviour of training processes with various quantisation values

How does the CMAC's generalisation level affect its approximation ability? It is observed that an increase in the generalisation level will usually lead to a decrease in the training error, but the trend is not obvious some times, as shown in Fig. 11 where three runs for a fixed quantisation combination=(0.075, 0.05) and three generalisation levels are depicted. However, high generalisation level will lead to slow convergence and a possible overfitting problem due to the excessive freedom of parameters.

In general, CMAC has two types of structural parameters, i.e., generalisation and quantisation. They determine the size of the CMAC model and the partitioning of the input space. For a given task, a general practice is to choose a small model that has enough capacity to fit the training data.

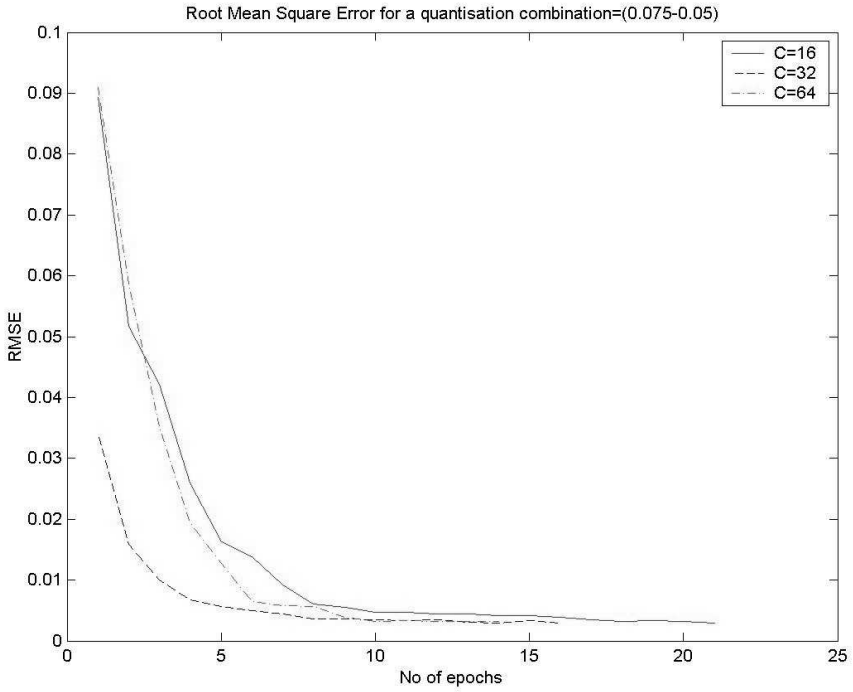

Figure 11 RMSE behaviour of training processes with various generalisation levels

3.2 Training Time

Comparing the training time of the CMAC model against that of the ANFIS model, it is very clear that the former has a much faster convergence. This can be seen in Table 1 where three ANFIS runs are depicted together with three CMAC runs (with the longest running times). The main reason is that the number of parameters updated at one time in CMAC is much less than in ANFIS.

Table 1 Training machine time of ANFIS and CMAC

RUN-ID	Parameter settings	Time (s)	Parameters updated at one time
CMAC-45	Quantisation=0.075-0.10, Generalisation=64	34.12	64
CMAC-33	Quantisation=0.05-0.10, Generalisation=64	170.34	64
CMAC-81	Quantisation=0.10-0.05, Generalisation=64	171.65	64
ANFIS-1	No. of membership functions=5	1061.09	105
ANFIS-2	No. of membership functions=8	9331.55	240
ANFIS-3	No. of membership functions=10	28399.37	360

Now let us examine the influence of the CMAC structural parameter settings on its training time. Training times with fixed generalisation level and varying quantisation values are shown in Fig 12. Training times with fixed quantisation values and varying generalisation levels are also shown in this figure. It is clear that with a higher generalisation level longer training time is required. However, it is not so clear about the relationship between the quantisation values and the training time. With smaller quantisation values, more receptive fields and associated memory (weights) will be generated, however, fewer epochs will be needed to reach the preset rmse value. These two effects will cancel each other someway in terms of the training time.

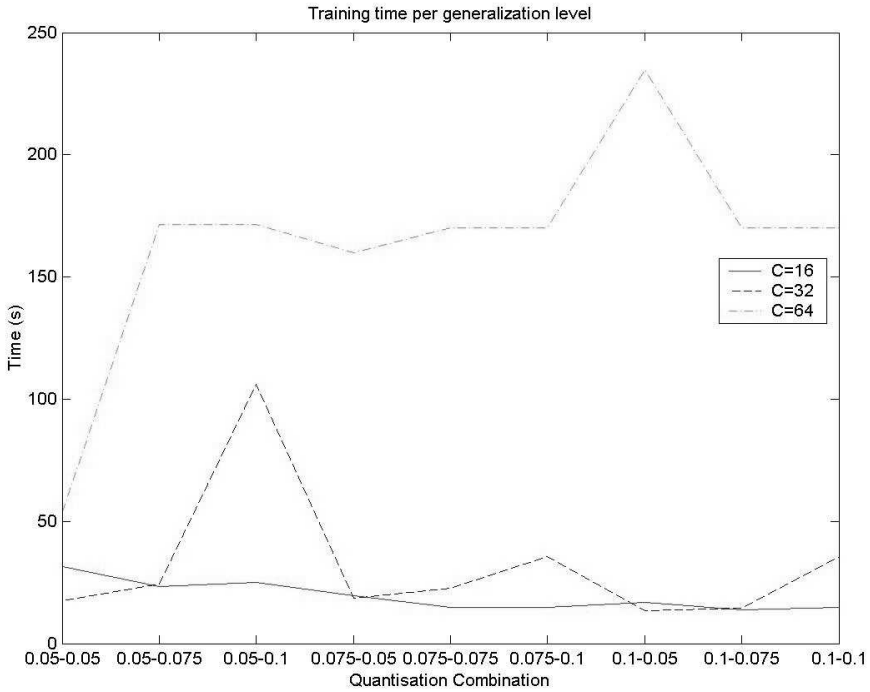

Figure 12 Training time of CMACs with various parameter settings

3.3 Memory Usage

It is quite interesting to notice the difference between the memory usages of the two neural networks. As discussed in section 2 about input space partitioning, both ANFIS and CMAC suffer from the curse of dimensionality problem. In general for the same task the number of required adjustable parameters in CMAC is larger than in ANFIS. However, there is only one RF in each layer activated by the input and only those weights associated with the activated RFs need to be updated. Therefore the active memory in CMAC is much less than in ANFIS.

Which model has better memory usage? As explained in section 2.2, both models are sensitive to the curse of dimensionality. In the ANFIS model, the number of free parameters is given by:

$$dof = m_1 \cdot m_2 \cdot n + (n+1) \cdot (m_2)^n \tag{4}$$

where m_1 is the number of free parameters in each fuzzy membership function (m_1=3 for a bell-shaped membership function), and m_2 is the number of membership functions for each input, and n is the input dimension. In the CMAC model, the number of receptive fields depends on the input dimension, and quantisation and generalisation values, which can be described by

$$NRF = \prod_{i=1}^{n} NI_i + (C-1)\prod_{i=1}^{n}(NI_i + 1) \tag{5}$$

where NRF is the number of receptive fields in total, $NI_i \approx (MaxValue_i - MinValue_i)/q_i$ is the number of intervals generated by the quantisation (q_i) on the ith input, with $MaxValue_i$ and $MinValue_i$ giving the range of the ith input, C is the generalisation, and n is the input dimension. For each receptive field, there is an associated weight. If each receptive field associates a unique weight, then the number of weights (free parameters) will be equal to the number of receptive fields. From (4) and (5), the number of receptive fields in CMAC is usually larger than the number of free parameters in ANFIS. However, a hashing function is used in CMAC to associate a receptive field to one of the smaller number of weights. Another measure used to combat the curse of dimensionality problem in CMAC is that only the weights associated with activated receptive fields will be

updated. As a result, the actual number of weights (memory used) is much smaller than the number of receptive fields and the number of weights updated at one time is fixed to *C*, which is usually much smaller than the number of free parameters in ANFIS. Some typical memory usages by CMAC and ANFIS models with various parameter setting are given in Table 2, which support the above analysis.

Table 2 Memory usages by CMAC and ANFIS models

RUN-ID	Parameter settings	*NRF* or *dof*	Memory used	Parameters updated at one time
CMAC-45	Quantisation=0.075-0.10, Generalisation=64	37584	2389	64
CMAC-33	Quantisation=0.05-0.10, Generalisation=64	55043	1912	64
CMAC-81	Quantisation=0.10-0.05, Generalisation=64	55043	1912	64
ANFIS-1	No. of membership functions=5	105	105	105
ANFIS-2	No. of membership functions=8	240	240	240
ANFIS-3	No. of membership functions=10	360	360	360

How do the structural parameters affect the CMAC's memory usage? To answer this question, Fig. 13 depicts the memory usages for a set of CMAC runs. It is observed that as the generalisation level increases the memory usage decreases. This can be explained as follows: firstly, although the number of receptive fields is very large with a high generalisation level, the number of active receptive fields at one time is determined by the generalisation level; Secondly, the number of overlapping receptive fields is greater when the generalisation level is higher, this means that for a smaller generalisation level more receptive fields are needed to well approximate a nonlinear function. It is also observed that as the quantisation increases, the memory usage decreases in general. This is due to the granularity produced by the quantisation. A smaller quantisation will produce a fine granular partition, resulting in more blocks in the intermediate mapping of the CMAC, therefore more receptive fields will be activated at the end of the training process.

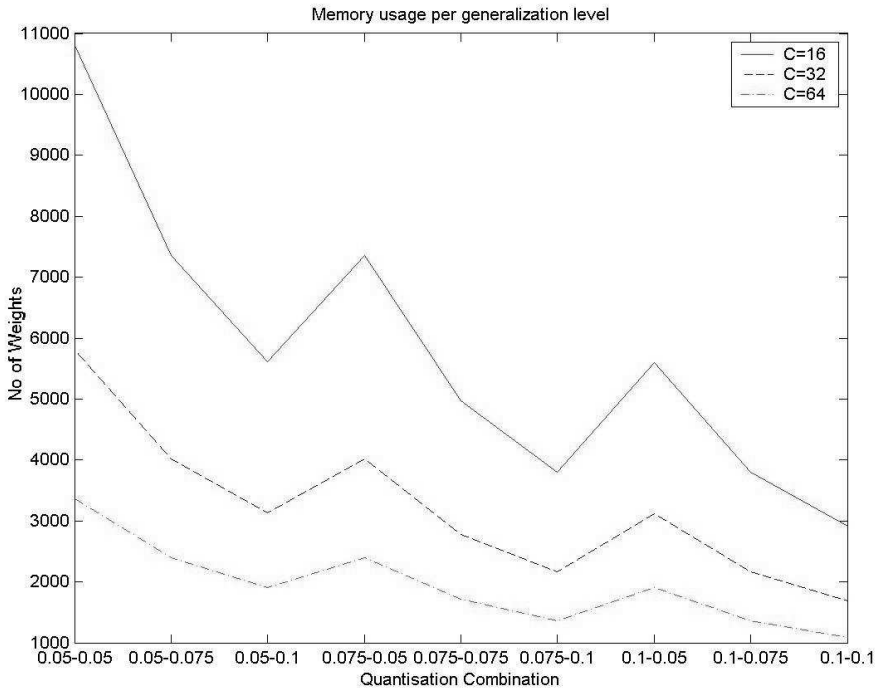

Figure 13 Memory usage by CMAC models

3.4 Discussion

An analysis of all experimental runs shows that, in general, smaller quantisation parameters (RF layer offsets) result in a finer input space partitioning, a larger number of receptive fields, more memory requirement, but not

necessarily longer training time. The influence of the generalisation parameter on approximation accuracy is not so explicit, but our experimental results show that larger generalisation parameters result in longer running time, but less memory requirement.

Fast training speed is an absolute advantage of the CMAC model, and it has comparable or even better approximation ability in comparison with ANFIS. A big advantage of the ANFIS model is that it can be directly applicable by well-developed linear techniques for state estimation and control as it can be treated as a local linear model. The idea of using hashing functions in CMAC could be useful for improving the training speed of ANFIS. Another idea of combining the advantages of CMAC and ANFIS models is to replace the constant weights in CMAC by linear functions of inputs. In this way, the CMAC model can also be interpreted as a local linear model, becoming therefore more applicable. It will be shown next how this addition influences the global approximation ability of the CMAC model.

4. CMAC with Linear Functional Weights

It was mentioned in the previous section that a possible modification to the CMAC model would be substituting constant weights by linear functions of the inputs. The output of the CMAC with linear functional (LF) weights will then be computed as:

$$y = \sum_{j=1}^{C} \frac{f(\delta_j(x_1,\cdots,x_n))}{\sum_{l=1}^{C} f(\delta_l(x_1,\cdots,x_n))} (w_{o,A_j} + w_{1,A_j}x_1 + ... + w_{i,A_j}x_i + ... + w_{n,A_j}x_n) \quad (6)$$

where $f(\delta_j(x_1,\cdots,x_n))$ represent receptive field functions, $\delta_j(x_1,\cdots,x_n)$ maps the input vector **x** into a RF field address in the *j*th layer of RFs, w_{i,A_j} are the adjustable weights, A_j represents the address of a set of adjustable weights, and *C* is the number of RF layers. The number of parameters to be updated will be now *(n+1)*C*, due to the addition of the excessive parameters. The learning rule for the modified CMAC can still be based on the LMS algorithm, some modifications should be derived though.

In order to analyse the effect of this modification, a similar function approximation experiment was conducted with the same training and testing data as used in the previous section for the CMAC with constant weights. The running conditions were as follows: a set of quantisation combinations (0.05, 0.075, 0.10) and three levels of generalisation (C=16, 32, 64) were considered; and the training process stopped when either the preset maximum number of epochs or root mean square error (rmse) of test-set validation is satisfied. In the experiment, the maximum number of epochs was set to 80 and rmse threshold set to 0.003. The learning rates have to be chosen carefully for the CMAC with LF weights, because the change in w_{0,A_j} has different influence on the model output from the change in w_{i,A_j}. In our experiment, different learning rates were used for updating w_{0,A_j} and w_{i,A_j} respectively. To guarantee the convergence, small learning rates or normalised learning algorithms should be applied. In the following subsections the CMAC with constant weights and the CMAC with LF weights are compared in terms of their approximation ability, training time, and memory usage. The results for the CMAC with constant weights and for the CMAC with LF weights are labelled as *CMACa-n* and *ECMAC-n* for easy understanding, respectively.

4.1 Approximation Ability

Will the CMAC with LF weights have powerful approximation ability, in comparison with the CMAC with constant weights? Typical approximation results by both the original and modified CMAC models are shown in Figs. 14 to 17. It can be concluded that both models have comparable approximation ability, but it is not clear which one is more powerful. An analysis of the RMSE behaviour of the learning processes shows that both models can reach a very small preset rmse value, but the CMAC with LF weights takes a longer time, as shown in Fig. 18.

As the training data here is from a non-smooth function and linear local models are generally good at modelling smooth functions, another experiment on modelling smooth nonlinear functions will be conducted in the next section to further investigate the approximation ability of the modified CMAC model.

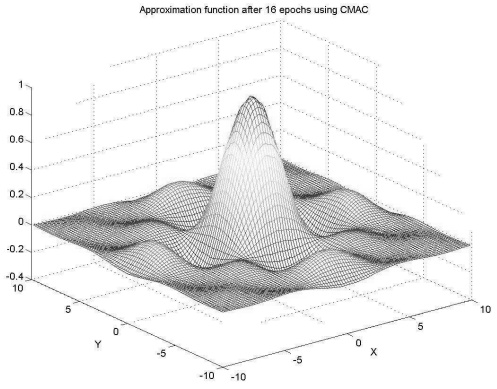

Figure 14 Approximation by CMAC with constant weights

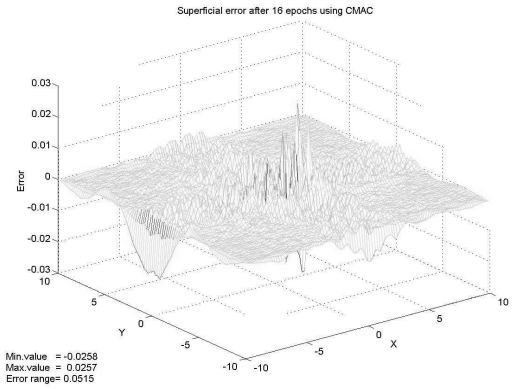

Figure 15 Error by CMAC with constant weights

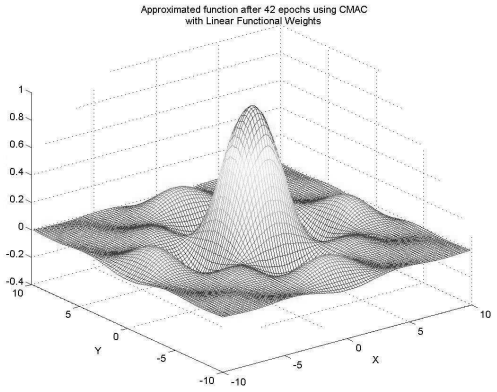

Figure 16 Approximation by CMAC with LF weights

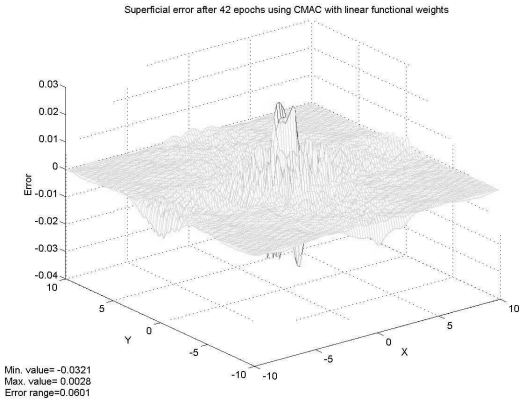

Figure 17 Error by CMAC with LF weights

4.2 Training Time

It is not expected to improve the training time by using linear functional weights in CMAC, intuitively the modification will increase training time as the number of free parameters is increased. However, the increase in the training time is reasonably small. In Table 3, the training times of both models with same structural parameter settings are depicted, which shows that the addition of linear functional weights does not produce a big impact on the training time. Fig. 18 has also shown the similar results about the training time.

Table 3 Training time of CMAC with constant and LF weights

RUN-ID	Parameter settings	Time (s)	No. of epochs
ECMAC-2	Quantisation=0.05-0.05, Generalisation=32	84.41	42
ECMAC-25	Quantisation=0.1-0.1, Generalization=16	68.81	45
ECMAC-16	Quantisation=0.075-0.10, Generalisation=16	74.37	51
CMACa-2	Quantisation=0.05-0.05, Generalisation=32	30.71	31
CMACa-25	Quantisation=0.1-0.1, Generalization=16	25.38	27
CMACa-16	Quantisation=0.075-0.10, Generalisation=16	31.12	25

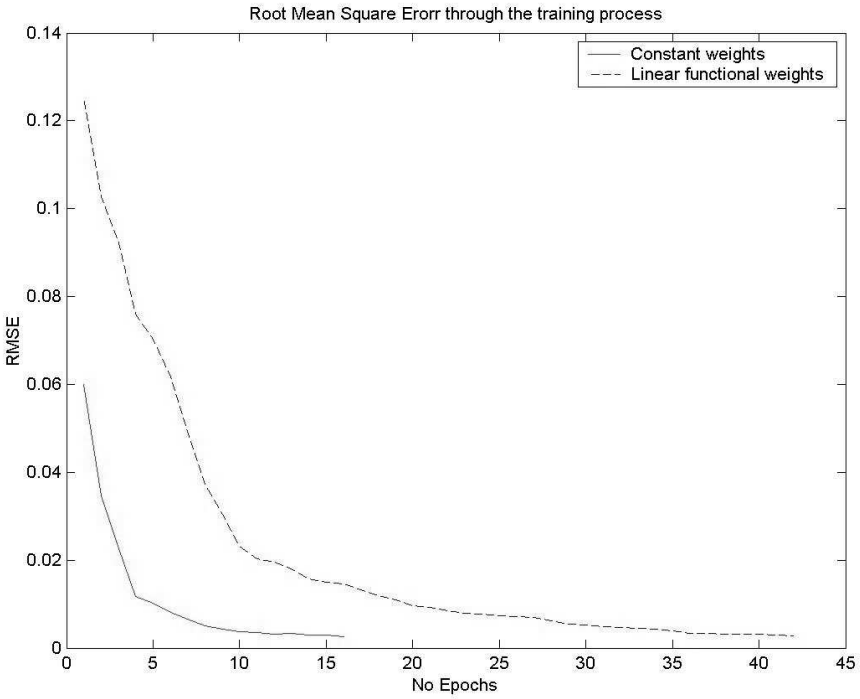

Figure 18 RMSE behaviour of CMAC training processes

4.3 Weight Distribution Analysis

The distributions of the 3 classes of weights, w_{0,A_j}, w_{1,A_j}, and w_{2,A_j}, are shown in Figs. 19-21, respectively. An analysis of the distribution of weight values shows that weights w_{o,A_j} have a distribution with most values in the range [–29.3665, 29.4507], much larger than w_{1,A_j} and w_{2,A_j}, and play a dominant role in the approximation. This is reasonable as the function modelled is highly non-smooth. In the next section, a smooth function is to be modelled and the role of the linear function part will be examined again.

4.4 Memory Usage

The memory usage of the CMAC with LF weights is proportional to the input dimension. Similar to the influence on the training time, the addition of linear functional weights increases the memory usage. However, compared to ANFIS, the number of parameters to be updated at one time in the CMAC with LF weights is still advantageous. The memory usages by the ANFIS, the CMAC with constant weights, and the CMAC with LF weights are given in Table 4 for comparison.

Table 4 Memory usages by various models

RUN-ID	Parameter settings	*NRF or dof*	Memory used	Parameters updated at one time
ECMAC-2	Quantisation=0.05-0.05, Generalisation=32	161133	17487	96
ECMAC-25	Quantisation=0.1-0.1, Generalisation=16	21045	8745	48
ECMAC-16	Quantisation=0.075-0.10, Generalisation=16	28080	11457	48
CMACa-5	Quantisation=0.05-0.05, Generalisation=32	53711	5815	32
CMACa-98	Quantisation=0.1-0.1, Generalisation=16	7015	2914	16

CMACa-62	Quantisation=0.075-0.10, Generalisation=16	9360	3805	16
ANFIS-3	No. of membership functions=10	360	360	360

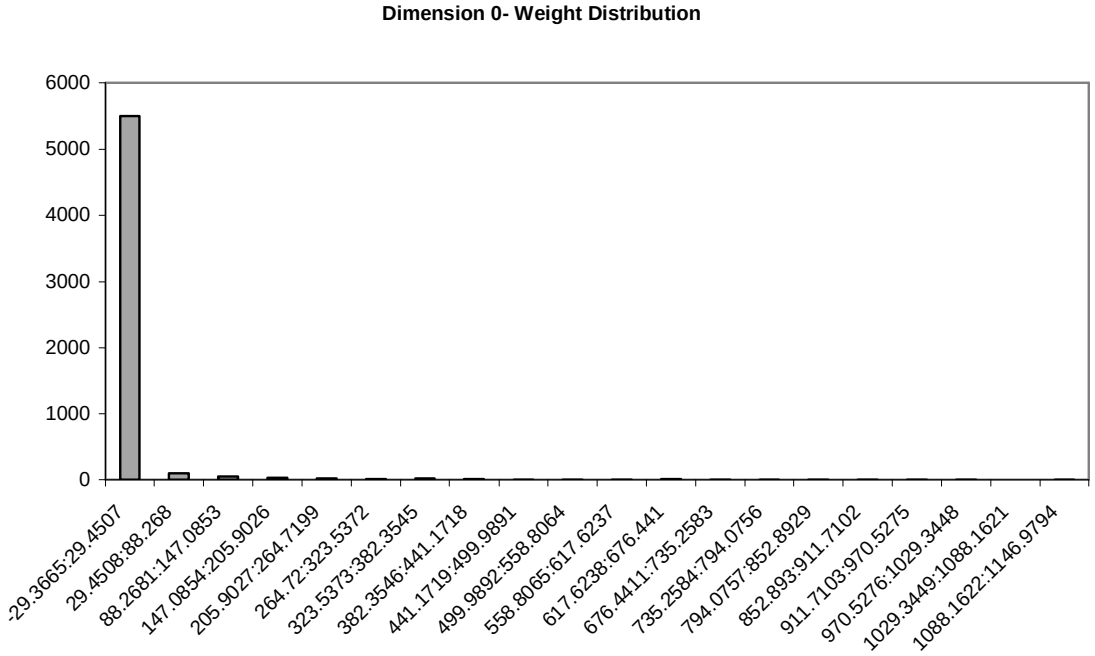

Figure 19 Distribution of weights w_{0,A_j}

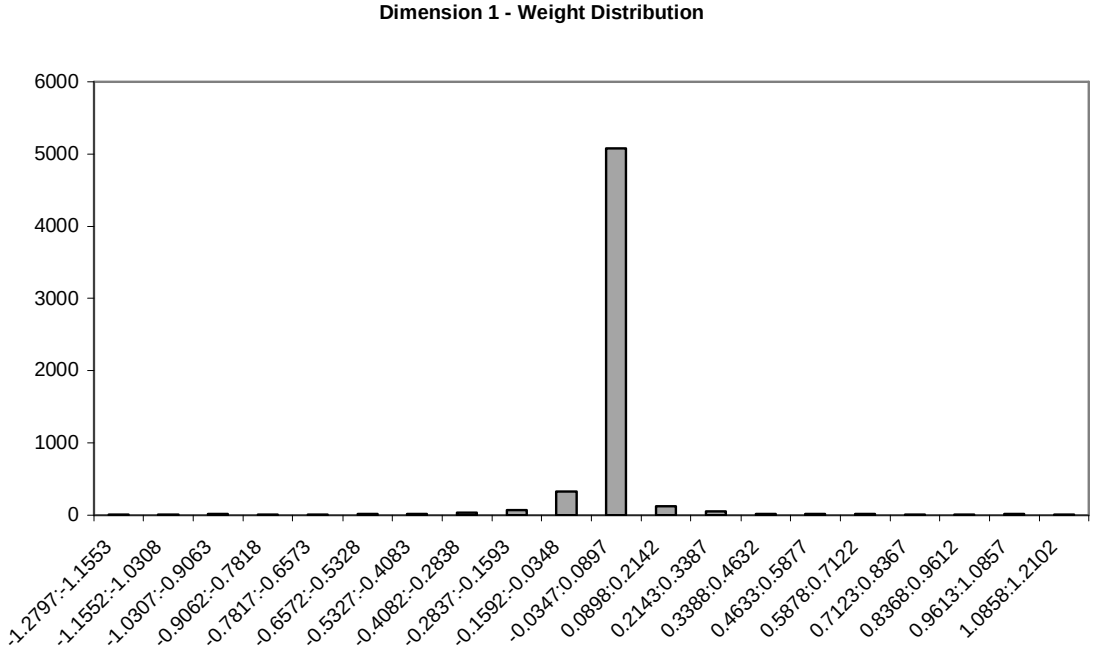

Figure 20 Distribution of weights w_{1,A_j}

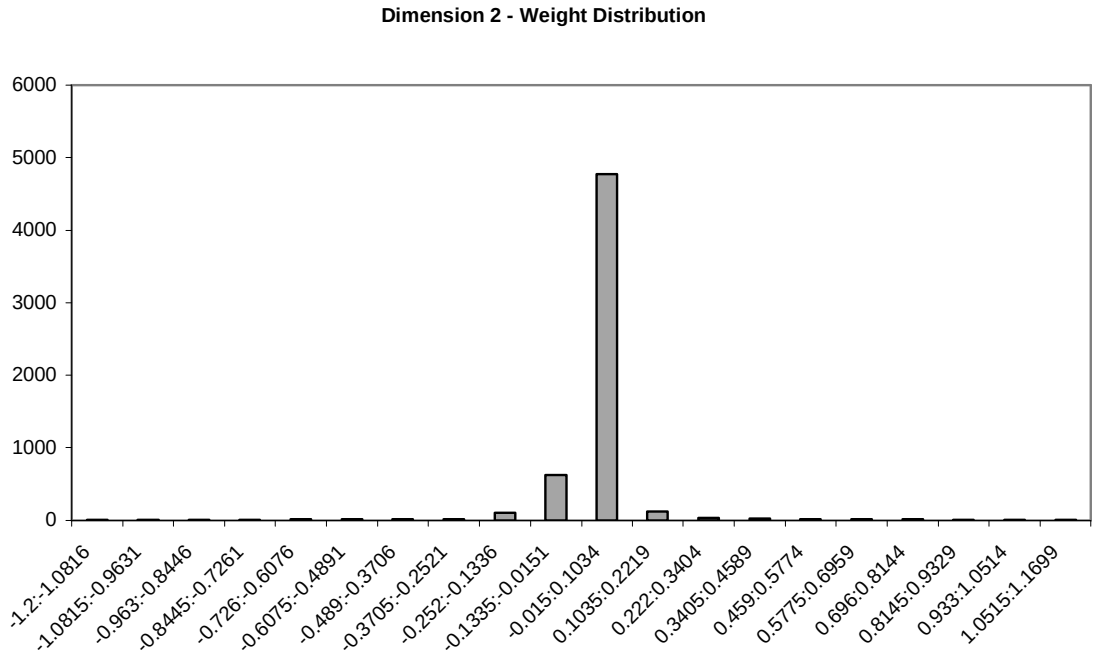

Figure 21 Distribution of weights w_{2,A_j}

4.5 Discussion

The experimental results in this section show that the CMAC with LF weights works very well, but the addition of linear local models can be pointless when modelling a non-smooth function in terms of its approximation ability, training time and memory usage. However, the CMAC with LF weights has an approximation ability that can match the ANFIS model, and at the mean time it has the advantage of fast training convergence. Next section will examine the properties of the CMAC with LF weights while modelling smooth nonlinear functions, giving more favourable results.

5. Investigation on Local Linearisation Approximation by the CMAC with Linear Functional Weights

5.1 Advantages of Linear Local Models

The main advantages of the Takagi-Sugeno fuzzy model, such as ANFIS, as Johansen noticed, stem from the fact that its consequence part is an affine dynamic model (or linear model) rather than a fuzzy set or constant value [16]. These advantages are:

- Using local linear models brings together fuzzy and conventional control theories.
- The relatively complex consequence part allows the number of fuzzy rules to be quite small in many applications.
- The model structure and local model properties can, in some applications, be easily related to the physics of the system, simplifying the model development and validation.

For many applications, it is very useful for a local linear model to be able to approximate the local linearisations of the nonlinear system to be modelled, i.e., in a local region nearby a given input point x_0, the local linear model should fit with in the linearisation of the nonlinear system: $y_{linearisation} = f(x_0) + f'(x_0)(x - x_0)$. The ability of the CMAC with LF weights to model local linearisations of smooth nonlinear systems will be investigated in this section.

5.2 Approximation of Smooth Nonlinear Functions

In order to examine the local linearisation approximation ability of the CMAC with LF weights, another experiment was set up to approximate the following function:

$$Y = X^2 + 1, \qquad -1 < X < 3 \tag{7}$$

For the purpose of visualisation of experimental results, a one-dimensional function is considered here. 40 training data points were generated with randomly chosen inputs with some noise added and 50 testing data points were generated with equal spacing, as shown in Figs. 22 and 23, respectively.

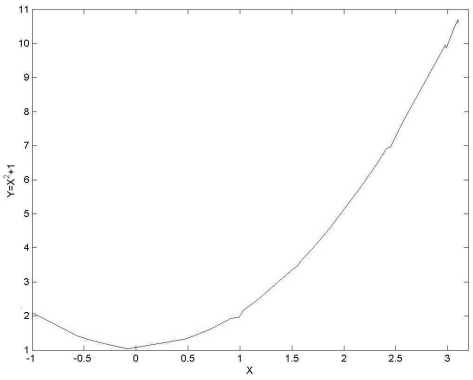

Figure 22 Training data

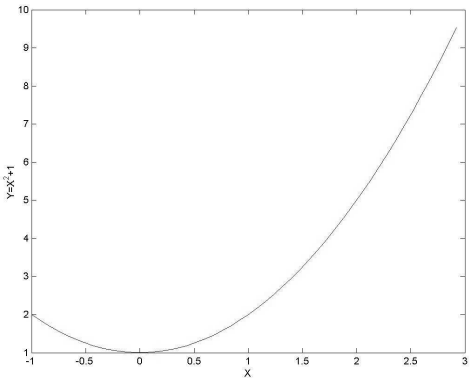

Figure 23 Testing data

The structural parameter settings were set as follows: the quantisation (q) equals 0.5, 0.75, 1.0, or 1.25, and the generalisation (C) equals 2, 4, 8, or 16. The maximum number of epochs was set to 80, and the maximum rmse allowed was set to 0.07. The experimental results of the CMAC with constant weights and the CMAC with LF weights will be compared in the following subsections.

5.2.1 Global Approximation

In this experiment of modelling smooth nonlinear functions, it was observed that the CMAC with LF weights outperforms the CMAC with constant weights in terms of their global approximation ability. The approximation errors of typical runs of the CMAC with constant weights and the CMAC with LF weights (with the same structural parameter settings) are shown in Table 5 and Table 6, respectively. As shown in Table 5, the CMAC with constant weights cannot achieve the preset RMSE of 0.07, while the CMAC with LF weights can.

Table 5 Approximation error by the CMAC with constant weights

RUN ID	q	C	RMSE	Minimum value	Maximum value	Size of error range
1b	0.5	2	0.089193	-0.2845	0.0406	0.3251
3b	0.5	8	0.088091	-0.2146	0.0986	0.3132

Table 6 Approximation error by the CMAC with LF weights

RUN ID	q	C	No of epochs	RMSE	Minimum value	Maximum value	Size of error range
1	0.5	2	72	0.060880	-0.1533	0.0686	0.2219
3	0.5	8	72	0.062266	-0.0955	0.1257	0.2212

Fig. 24 shows a comparison of the errors given by the two CMAC models with the same configuration (q=0.5, C=2), with the CMAC with constant weights depicted by a solid line and the CMAC with LF weights by a dashed line. It can be observed that the CMAC with constant weights produces larger errors.

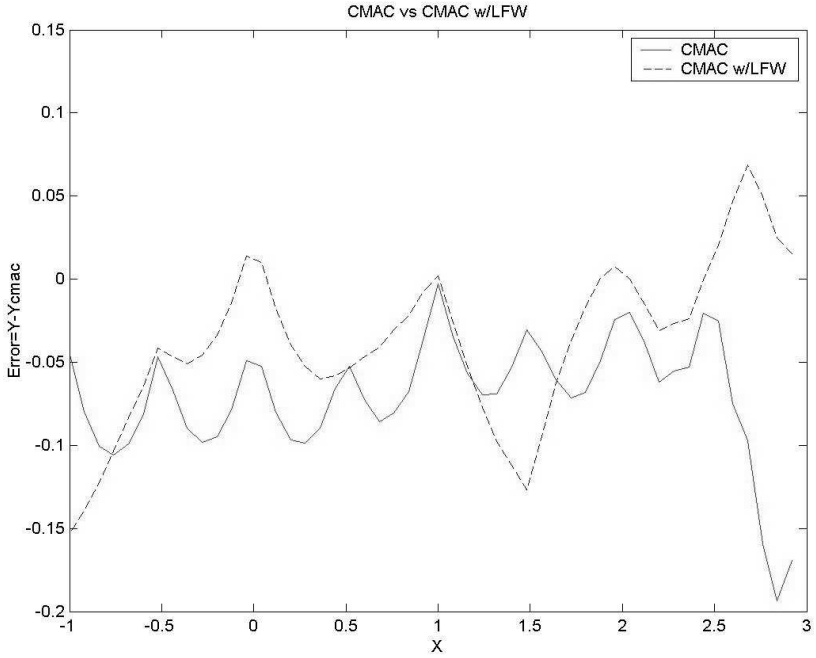

Figure 24 A comparison of errors by CMAC models with and without LF weights

Although some parameter settings achieve similar approximation results, different structural parameter settings lead to different training process. Fig. 25 shows the RMSE behaviours in training the CMAC with LF weights, with parameter setting {q=1.25 and C=2} giving better performance than the other settings. The network output with the best parameter setting is shown in Fig. 26, in comparison with the real function.

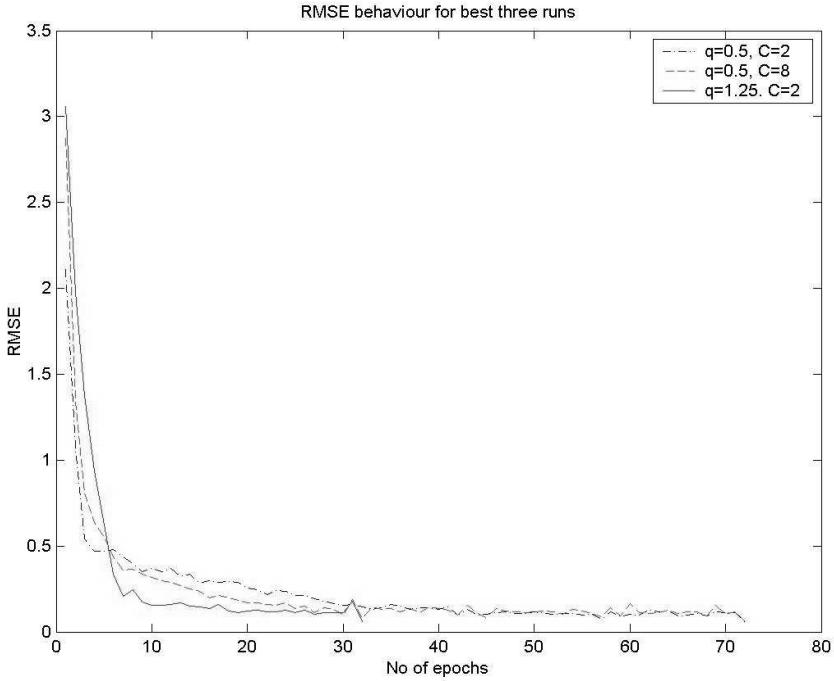

Figure 25 RMSE behaviours in training the CMAC with FL weights

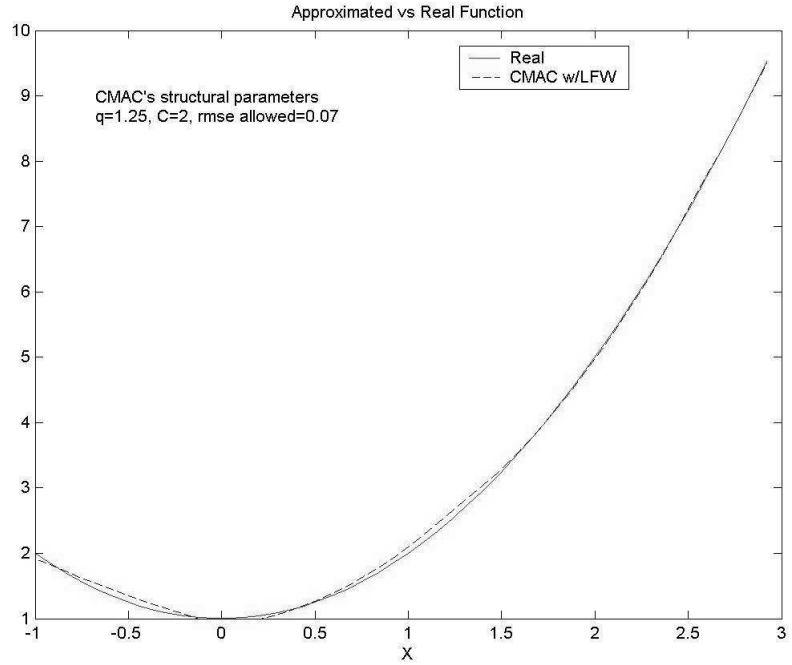

Figure 26 Output of the CMAC with LF weights in comparison with the real function

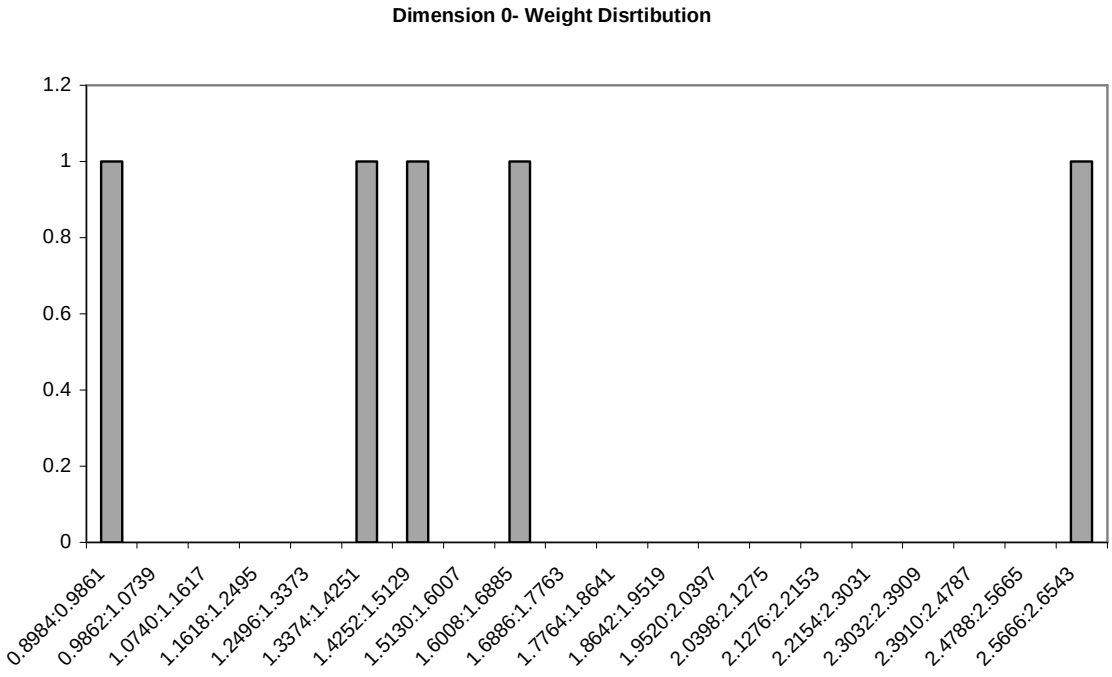

Figure 27 Distribution of weights w_{0,A_j}

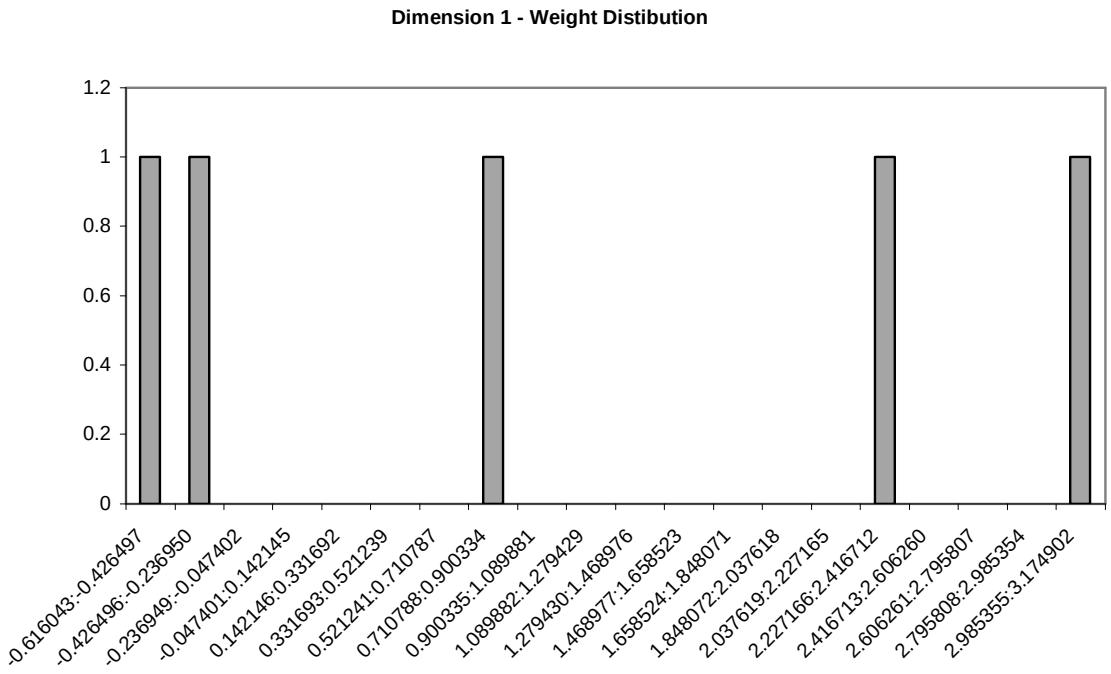

Figure 28 Distribution of weights w_{1,A_j}

5.2.2 Weight Distribution Analysis

In the experiment of modelling non-smooth functions, it was observed that the values of w_{0,A_j} are much larger than those of w_{i,A_j}, that is, weights w_{0,A_j} play a dominant role in the approximation. In a similar way, the distributions of the 2 classes of weights, w_{0,A_j} and w_{1,A_j}, for modelling the smooth nonlinear function are shown in Figs. 27 and 28, respectively. ***REMARKS***

5.2.3 Local Linearisation Approximation

At a given input x_0, the local linearisation of the function $Y=X^2+1$ is given by

$$Y_{linearisation} = (2x_0)X + (1 - x_0^2) \tag{8}$$

and the linear local model of the CMAC is:

$$Y_{CMAC} = aX + b \tag{9}$$

where

$$a = \sum_{l=1}^{C} \frac{f(\delta_j(x_0))}{\sum_{l=1}^{C} f(\delta_l(x_0))} \cdot w_{1,A_j} \tag{10}$$

$$b = \sum_{l=1}^{C} \frac{f(\delta_j(x_0))}{\sum_{l=1}^{C} f(\delta_l(x_0))} \cdot w_{0,A_j} \tag{11}$$

If the CMAC model produces good local linearisation approximation, Y_{CMAC} should match $Y_{linearisation}$ very well. Now consider the CMAC with LF weights and the parameter setting of {q=1.25, C=2}. Assuming x_0=2.5, then the values of weights and receptive functions can be calculated as shown in Table 7. From (8)-(11), the real local linearisation at this point is $Y_{linearisation}=5X-5.25$, and the local linear model given by the CMAC is $Y_{CMAC}=2.373X+1.354$. Other input points have also been examined in a similar way. The real local linearisations

(dashed lines) and local linear models (dash-dot lines) at three different points are shown in Fig. 29. It is clear that although the CMAC model does produce good global approximation, but not good local linearisation approximation.

Table 7 Values of weights and RF functions at x_0=2.5

j (j=1,…,C)	$f(\delta_j(x_0))$	w_{0,A_j}	w_{1,A_j}
1	1	2.65442012	3.17490254
2	128	1.34338778	2.36652879

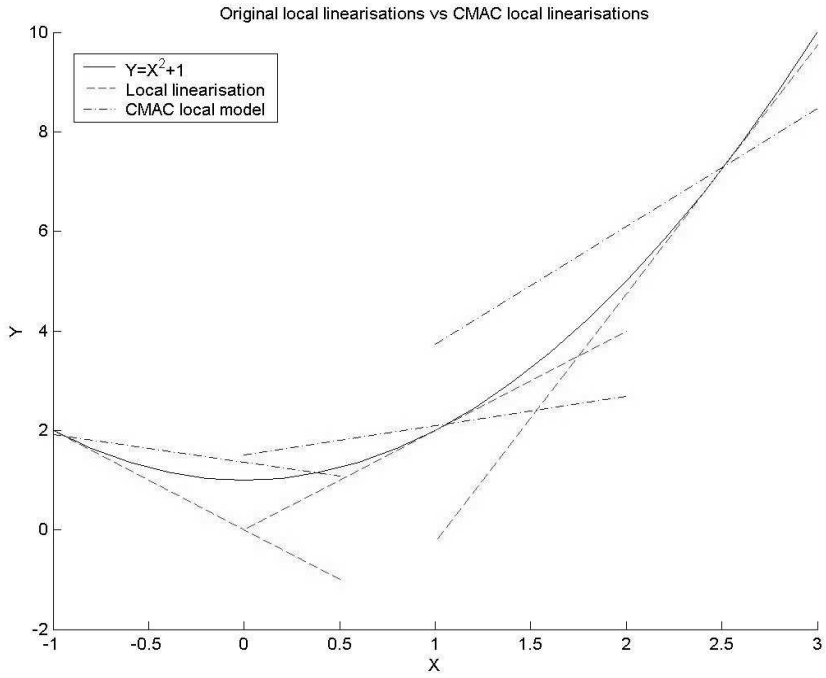

Figure 29 Local linearisations of the real function (dashed lines) and the CMAC model (dash-dot lines)

5.2.4 Discussion

In modelling smooth nonlinear functions, the CMAC with LF weights outperforms the CMAC with constant weights in terms of their approximation ability. However, the CMAC models do not satisfy Johansen's second requirement for model approximation: *It is sometimes required and often desirable that the local linear models are accurate approximations to the local linearisations of the real function* [16]. A possible solution to this could be the regularisation technique that penalises the difference between the first derivatives of the model and the real function. This will be further investigated in our future research.

6. Conclusions

A comparison between ANFIS and CMAC models was conducted in this paper, in terms of input-space partitioning, curse of dimensionality, approximation ability, training time, and memory requirement. Both ANFIS and CMAC models have very good approximation ability, but CMAC models have an obvious advantage of fast training speed over ANFIS models. The main advantage of ANFIS is the good interpretability and applicability due to the use of linear local models. A CMAC modification has been proposed by introducing linear functional weights such that the modified CMAC model is a local linear model. It has been shown that in modelling non-smooth nonlinear functions the modification may be useless, but in modelling smooth nonlinear functions the CMAC with LF weights outperforms the CMAC with constant weights. It is expected that the modified CMAC, with the advantage of fast training speed and the potential to approximate local linearisations

of nonlinear functions, would find wide applications in nonlinear modelling, state estimation and control (e.g., robot arm modelling and control). As far as the applications are concerned, it is highly desired that the local linearisation approximation ability of the CMAC with LF weights would be improved by using regularisation techniques.

References

[1] J.S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", *J. of Dynamic Sys., Measurement and Control, ASME*, pp. 220-227, Sep 1975.

[2] J.S. Albus, "Data Storage in the Cerebellar Model Articulation Controller (CMAC)". *J of Dynamic Sys., Measurement and Control, ASME*, pp. 228-233, Sep 1975.

[3] J.S. Albus, "A Model of the Brain for Robot Control Part 1: Defining Notation", *Byte Magazine*, Vol. 4, No. 6, pp. 10-32, June 1979.

[4] J.S. Albus, "A Model of the Brain for Robot Control Part 2: A Neurological Model", *Byte Magazine*, Vol. 4, No. 7, pp. 54-95, June 1979.

[5] J.S. Albus, "A Model of the Brain for Robot Control Part 3: A Comparison of the Brain and our Model", *Byte Magazine*, Vol. 4, No. 8, pp. 66-80, Aug 1979.

[6] Jyh-Shing Roger Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System", *IEEE Trans. Sys., Man and Cybernetics*., Vol. 23, No. 3, May/June 1993.

[7] T. Kavli, "ASMOD –An Algorithm for Adaptive Spline Modelling of Observation Data", *Int. J. of Control*, Vol. 58, No. 4, pp. 947-967, 1999, Oct 1993.

[8] W.T. Miller and F.H. Glanz, (Nov. 2001), "UNH_CMAC Version 2.1. The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer – CMAC", [Online] Available: http://www.ece.unh.edu/robots/cmac.htm.

[9] W.T. Miller; R.P. Hewes; F.H. Glanz and L.G. Kraft, "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural Network-Based Learning Controller", *IEEE Trans. Robotics and Automation*, Vol. 6, No. 1, pp 1-9, Feb 1990.

[10] W.T. Miller; F.H. Glanz and L.G. Kraft, "CMAC: An Associative Neural Network Alternative to Backpropagation", *IEEE Proceedings*, Vol. 78, No. 10, pp 1561-1567, Oct 1990.

[11] E. Pak-Cheung, *An Improved Multi-dimensional CMAC Neural network: Receptive Field Function and Placement*, PhD Thesis, Dept. Electrical and Computer Engineering, New Hampshire Univ., New Hampshire, USA, 1991.

[12] O. Nelles, *Nonlinear System Identification with Local Linear Neuro-Fuzzy Models*, Ph.D. Thesis, Darmstadt University, 1999.

[13] Q. Gan and C.J. Harris, "Fuzzy local linearization and local basis function expansion in nonlinear system modeling", *IEEE Trans. on Sys. Man and Cybernetics*, Vol. 29B, No 4, pp. 559-565, 1999.

[14] Q. Gan and C.J. Harris, "A hybrid learning scheme combining EM and MASMOD algorithms for fuzzy local linearization modeling," *IEEE Transactions on Neural Networks*, Vol. 12, No. 1, pp. 43-53, 2001.

[15] X. Benavent Garcia, *Dynamic Systems Modelling and Control using Neural Networks and Linear Models: Application to a Simulation Platform Control*, PhD Thesis, Dept. Informatics, Faculty of Physics, University of Valencia, Valencia, Spain, 2001.

[16] T.A. Johansen, R. Shorten and R. Murray-Smith, "On the Interpretation and Identification of Dynamic Takagi-Sugeno Fuzzy Models", *IEEE Trans. Fuzzy Systems*, Vol. 8, No 3, pp 297-313, June 2000.