Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

Online Learning for Hierarchical Networks of Locally Arranged Models using a Support Vector Domain Model

Florian Hoppe and Gerald Sommer

***Abstract*— We propose two new developments for our supervised local linear approximation technique, the so called *Hierarchical Network of Locally Arranged Models*. A new model will be presented that defines those local regions of the input space in which linear models are trained to approximate the target function. This model is based on a one-class support vector machine and helps to improve the approximation quality. Secondly, an online learning algorithm for our approach will be described that can be used in applications where training data is only available as a continuous stream of samples. It allows to adapted a network to a function that may change over time. The success of these two developments is proven with three benchmark tests.**

I. Introduction

Supervised local (or also called piecewise) linear approximation (LLA) methods (such as [16], [10], [6]) use a set of linear models to approximate a non-linear target function. These approaches follow the divided and conquer strategy to achieve good global approximation by the right combination of good local linear models. These models are trained in a way that they can only approximate the target function in a local region of the input space. Such a local region is commonly called the domain of a linear model.

The different LLA approaches have to answer two basic questions: How is the domain of a local model defined? And, how is the input space split up into a set of such domains? An answer to the first question conditions the possible shapes of the local region that is governed by a linear model. If a domain model is very flexible, it is more likely that the corresponding linear model can approximate the target function perfectly. Otherwise more linear models are required to divide the input space on a finer scale to realize similar approximation quality. Hence, the shapes of the domains i.e. its model strongly influences the needed number and the possible approximation performance of the linear models in a network. On the other hand side, a domain model should not be too complicated as this can become computational too demanding: the needed number of parameters should be kept small so that their estimation remains feasible. So, it is the goal of LLA approaches to find a good compromise between a flexible but still simple domain model.

Given such a model, one can decide on a strategy to divide the input space in order to answer the second question. The needed process should meet two main goals: the achieved approximation performance of the linear models should be as good and their number as small as possible. Since these two goals are strongly dependent on each other, again a good compromise has to be established to fulfill both.

Florian Hoppe and Gerald Sommer are with the Department of Cognitive Systems, Institute of Computer Science, Christian Albrechts University, Kiel, Germany (email: {fh,gs}@ks.informatik.uni-kiel.de).

II. State of the Art LLA Techniques

The k-nearest neighbor (k-NN) approach (e.g. [4]) is a straightforward realization of the local linear model idea. It solves for each input $x \in \mathbb{R}^n$ the least squares problem:

$$\min_{\alpha(x),\beta(x)} \sum_{i=1}^{k} \Big(y_i - \alpha(x) - \beta(x) x_i \Big)^2, \tag{1}$$

where the training samples (x_i, y_i) with $i = 1, \ldots, k$ are the k-nearest neighbors to the input x. The set of nearest neighbors is determined by comparing the distances between the input x and all samples from the training set w.r.t. some metric (typically, the Euclidean distance). The estimated output $\hat{y} \in \mathbb{R}^m$ is consequently given by $\hat{y} = \alpha(x) + \beta(x)x$. Note, that the k-NN approach is non-parametric and has no explicit domain model since it uses the whole training set to compute an output value. Hence it is very inefficient in both memory and computational costs.

In contrast to that, approaches like [16], [10], [6] can be formalized as:

$$\hat{y}(x) = \sum_{k=1}^{M} g_k(x) \beta_k^T \tilde{x}, \tag{2}$$

where M is the number of used linear models with their coefficient $\beta_k \in \mathbb{R}^{n+1}$ and $\tilde{x} = (x, 1)^T \in \mathbb{R}^{n+1}$ is the extended input x that allows a constant term in the linear equation. With the weighting function $g_k(\cdot) \in \mathbb{R}$ the domain of the k-th linear model is defined. This definition is specific to the different approaches. Their common idea is that depending on the distance of the input x to the domain, the weighting factors are larger or smaller and hence, will weight the output $\beta_k^T \tilde{x}$ of the linear models differently. The typical definition of the weighting functions $g_k(\cdot)$ is based on a radial basis function (RBF). The resulting domains have the shape of hyper-ellipsoids in the input space. Depending on the approach specific definition of the weighting functions these hyper-ellipsoids have different degrees of freedom. E.g. in [10] the main axis of the hyper-ellipsoids are restricted to be parallel to the axis of the input space, while in [6] these can be oriented in any direction.

Besides differences in the definition of the domain model the approaches vary more importantly in their strategies to split up the input space. In [10] the position and size of the domains are changed with Hebbian adaptation steps and by a gradient decent approach to minimize the least squares error function over the training set. The authors of [16] prefer to

minimize a so-called locally weighted error function which emphasizes that training samples only effect the domain's parameter they belong to. Our own offline learning algorithm (introduced in [5] and extended in [6], [7]) is a recursive scheme that divides by means of a special clustering process the input space into single domains. This divide and conquer strategy is repeated until the required approximation quality is achieved in each domain or no more data samples are available in a local region to divide it furthermore.

In the following, we will present two new developments in our LLA the so called *Hierarchical Network of Locally Arranged Models* (HLAM) approach. A new domain model is described that allows more flexible shapes than those of a hyper-ellipsoid. More important an online learning algorithm for our network was realized. It is designed to be used in an application where training data is only available as a continuous stream of samples. The idea is that the input space of the target function is being explored during the runtime of the implemented system. Furthermore a HLAM should capture dynamic changes of the target function. This is different to the offline learning scenario where a certain input value always corresponds to a certain output value. The online learning algorithm can adapted a HLAM to a function that may change over time.

III. New Developments for Hierarchical Networks of Locally Arranged Models

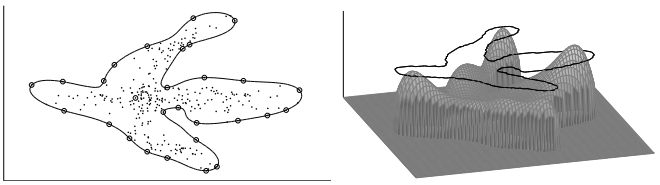
The basic definition of a HLAM is given with (2). The weighting functions are defined as:

$$g_k(x) = \begin{cases} 1 & : \quad k = \arg_i \max a_i(x) \\ 0 & : \quad \text{else} \end{cases} , \tag{3}$$

where $a_i(\cdot)$ with $i = 1, \ldots, M$ are so called activation functions of the *M* linear models (their definition will be given below). So, in the HLAM approach only one linear model is exclusively selected to compute the output of the whole network. Only the output of the linear model with maximal activation value will be weighted with one, all the other weights will be set to zero. This exclusive selection stands in contrast to the other LLA approaches which mix the output of different models. Our definition promotes that the linear models are strictly specialized for their domain. This is supported by our learning algorithms as these train the linear models only with the data from their own domains. This parameter estimation is done individually for each linear model with a standard least squares method (e.g. [4]). Other approaches determine the parameters of the linear models with all available data using weighted least squares schemes.

The Support Vector Domain Model

The definition of the activation functions are based on a special distance function that defines the shape of a domain. In [6] these functions were basically radial basis functions defining domains shaped like hyper-ellipsoids. As an alternative we propose to use a one-class support vector machine (SVM) to define the distant function $d_k : \mathbb{R}^n \to \mathbb{R}$ as:



(a) (b)

Fig. 1. Example of a support vector domain. Panel (a) shows a set of data samples and the domain boundary realized with a radial basis SVD. These samples that are support vectors are marked with circles. Panel (b) shows the corresponding activation function.

$$d_k(x) = \sum_{i=1}^{S_k} \alpha_{k,i} K(s_{k,i}, x), \tag{4}$$

where $S_k \in \mathbb{N}$ is the number of the support vectors $s_{k,i} \in \mathbb{R}^n$ with their weights $\alpha_{k,i} \in \mathbb{R}$ and $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a kernel function. The user can choose any kernel function. Popular options are:

$$\begin{aligned} K(x,y) &= (1 + x^T y)^d, && \text{(polynomial)} \\ K(x,y) &= \exp(-\tfrac{1}{\sigma}\|x-y\|^2), && \text{(radial basis)} \\ K(x,y) &= \tanh(\kappa_1 x^T y + \kappa_2). && \text{(sigmoidal)} \end{aligned} \tag{5}$$

Given the function $d_k(\cdot)$ of the *k*-th domain, its activation functions is defined with:

$$a_k(x) = \begin{cases} d_k(x) & : \quad d_k(x) \geq \gamma_k \\ -\infty & : \quad \text{else} \end{cases} , \tag{6}$$

where the threshold γ_k is a scalar parameter. The activation value is equal to the value of (4) if the threshold is met or exceeded. This draws a sharp boundary around the domain which strictly separates the domain's inside from its outside. Figure 1 shows an example of such a support vector domain (SVD). In the following all the parameters of the *k*-th domain will be denoted with $\Phi_k = (s_{k,1}, \ldots, s_{k,S_k}, \alpha_{k,1}, \cdots, \alpha_{k,S_k}, \gamma_k)$.

The support vectors and the weights of a domain are determined with a standard method for one-class support vector machines (see e.g. [14]) for a set of input samples $\{x_i\}$. This set is selected by the HLAM learning algorithm which ensures that the samples belong to a local region of the input space. Given the support vectors and their weights the threshold γ_k is set to the minimum value of the function $d_k(\cdot)$ of all support vectors $s_{k,i}$, i.e. $\gamma_k = \min_{s_{k,i}} d(s_{k,i})$. This establishes a domain boundary that encloses all the training data.

Note, that the user has to specify additional parameters (the regulation parameter ν and e.g. for a radial basis kernel its width σ) for the employed SVM learning algorithm and the chosen kernel function. These parameters have to be manually optimized for a specific data set.

The thresholds γ_k have two purposes in the HLAM approach. On one hand side, it allows to realize an automatic outlier rejection mechanism. A data sample with input values that are atypical for a specific application can be detected by simply testing if the sample belongs to any domain.

If not, an exception can be thrown by the program that the input acquisition method produced data that can not be handled by the trained network. This mechanism increases the reliability of a system that employs a HLAM. Standard machine learning technique do not offer this feature as they always calculate some output regardless how invalid the input was. If such an outlier rejection mechanism is not needed all thresholds γ_k should be set to minus infinity. Then this local model will compute the output which domain is closest to the input, regardless if this belongs to the domain or not. This kind of nearest neighbor selection suits well the basic idea of the HLAM approach that assigns linear models to local regions of the domain space.

On the other hand, the sharp domain boundary defined by the threshold γ can be exploited for a novelty detection. It depends on the view point i.e. on the task at hand if a data sample should be regarded as an invalid input or if the same sample is taken as a new piece of information. In the first case the sample should not be processed as it may cause harmful effects. In the latter it could be used to extend a trained HLAM. This idea is utilized in the new online learning algorithm.

The advantage of the new support vector domain model is the great flexibility of its boundary. In contrast to the former hyper-elliptical domain, the SVD model can enclose non-convex regions of the input space because the kernel functions can induce complex non-linear boundaries. Hence a given set of training samples can be more tightly enclosed. This ensures that only those regions of the input space belong to domains that really contain data samples. The positive effect is that the corresponding linear models are only responsible for those local regions where they received training data. Hence they can be better specialized to their domains. That should improve the approximation performance of the whole network. Furthermore, data samples which may be widespread in the domain space can be adequately enclosed by one single support vector domain and hence approximated by one linear mode. Since in such a case more data is available to train a linear model its approximation performance should be improved. Furthermore, with larger domains HLAMs become smaller. If more data can be assigned to the single domains, their number will be reduced.

The Online Learning Algorithm

Two basic ideas ground the development of the online learning algorithm. The activation function of a domain defines the responsibility of the corresponding linear model to a given training sample. This information is used to decide which linear model should be adapted to the new training sample. On the other hand, a so called neighborhood graph is used to divide the domain space into local regions w.r.t. established domains. This is important to decide where new domains should be established. The pseudo code is given in Algorithm 1. It defines the procedure `UpdateHLAM` that should be called when a new sample (x, y) is available.

The algorithm works as follows: A HLAM is initialized with no local model at all. The algorithm has a meta-

Algorithm 1: Pseudo code of online learning algorithm.

Function `UpdateHLAM`
Input : $(x, y), M^t = \{(\beta_k^t, \Phi_k^t, T_k^t)\}, B^t$
Output: $M^{t+1} = \{(\beta_k^{t+1}, \Phi_k^{t+1}, T_k^{t+1})\}, B^{t+1}$
begin
  if $M = \emptyset$ **then**
    $B^{t+1} \leftarrow B^t \cup \{(x, y)\}$
    if $|B^{t+1}| \geq$*minSamples* **then**
      $(\beta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModel`$(B^{t+1})$
      $M^{t+1} = (\beta^{t+1}, \Phi^{t+1}, B^{t+1})$
      $B^{t+1} = \emptyset$
  else
    Err$\leftarrow \min_k L(\tilde{x}^T \beta_k^t, y)$
    if *Err* $\leq$*maxError* **then**
      $l \leftarrow \arg_k \max d_k(x)$
      $(\beta_l^{t+1}, \Phi_l^{t+1}, T_l^{t+1}) \leftarrow$`UpdateM`$(T_l^t, (x, y))$
    else
      $l \leftarrow \arg_k \max a_k(x)$
      if $d_l(x) \geq \gamma_k$ **then**
        $(\beta_l^{t+1}, \Phi_l^{t+1}, T_l^{t+1}) \leftarrow$`UpdateM`$(T_l^t, (x, y))$
      $B^{t+1} \leftarrow B^t \cup \{(x, y)\}$
      $G \leftarrow$`GetNeighborhoodGraph`$(B^{t+1}, \{\Phi_k^t\})$
      $(l, s) \leftarrow \arg_{i,j} \max G_{i,j}$
      if $G_{l,s} \geq$ *minSamples* **then**
        $T^{t+1} \leftarrow$`GetSamples`(B^{t+1}, l, s)
        $B^{t+1} \leftarrow$`RemoveSamples`(B^{t+1}, l, s)
        $(\beta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModel` (T^{t+1})
        $M^{t+1} \leftarrow M^t \cup (\beta^{t+1}, \Phi^{t+1}, T^{t+1})$
end
Function `UpdateModel`
Input : $T^t, (x, y)$
Output: $(\beta^{t+1}, \Phi^{t+1}, T^{t+1})$
begin
  $T^{t+1} \leftarrow T^t \cup \{(x, y)\}$
  if $|T^{t+1}| >$ *maxBufferSize* **then**
    $T^{t+1} \leftarrow$ `RemoveOldestSample`(T^{t+1})
  $(\beta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModel` (T^{t+1})
end
Function `GetNeighborhoodGraph`
Input : $T = \{(x_j, y_j)\}, \{\Phi_l\}$,
Output: $G \in \mathbb{N}^{M \times M}$
begin
  $G \leftarrow 0$
  forall j **do**
    $k_1 \leftarrow \arg_l \min \|x_j - \frac{1}{S_l} \sum_{i=1}^{S_l} s_{l,i}\|$
    $k_2 \leftarrow \arg_{\{l|l \neq k_1\}} \min \|x_j - \frac{1}{S_l} \sum_{i=1}^{S_l} s_{l,i}\|$
    $G_{k_1,k_2} \leftarrow G_{k_1,k_2} + 1$
    $G_{k_2,k_1} \leftarrow G_{k_2,k_1} + 1$
end

TABLE I

RESULTS OF THE MACKEY-GLASS BENCHMARK TEST (SEE TEXT FOR AN EXPLANATION ABOUT THE HLAM MARKED WITH A STAR).

Method	Number of Hidden Units	RMSE	
		Train Set	Test Test
HALM offline	100	0.0006	0.0015
HLAM online	123	0.0046	0.0048
HLAM offline*	39	0.0040	0.0043
GMN [10]	7	0.0100	0.0091
RBF-AFS [2]	21	0.0158	0.0128
OLS [1]	132	0.0107	0.0163

parameter *minSamples* that specifies how many samples at least have to be assigned to one linear model. After initialization, this user chosen number of samples have to be collected before the first linear model and its domain will be added to the HLAM. Therefore new samples are temporary stored in a buffer T^t. Throughout the runtime of the system, this buffer will contain all the samples that could not be assigned to a linear model of the HLAM. If the size of buffer T^t equals *minSamples*, the first model and its domain will be established with all the sample from the buffer.

The model's parameters β^t are estimated with a standard least squares method, while the domain's parameter Φ^t are determined as described above. In the pseudo code this both is encapsulated in the procedure `TrainModel`. Along with β^t and Φ^t an extra buffer T^t containing all the used training samples is added to the HLAM. This buffer T^t represents the sample history of each local model. How it works in detail will be explained down below.

After the first model was added to a HLAM, for each new training sample it will be decided if an already existing linear model should be updated or a new model should be added to the HLAM. The criterion for this decision is if the sample could be successfully approximated or not. Therefore the minimal loss $L(\tilde{x}^T\beta_k^t, y)$ of all linear models of a HLAM is determined and compared with a user chosen threshold *maxError*. This loss function can be selected as appropriate. We use: $L(\hat{y}, y) = (\hat{y} - y)^2$.

If the approximation performance is good enough, the new sample is used to update one linear model. Before the question which model should be updated can be settled one should first discuss how this is done: As noted above, the online algorithm stores for each linear model the history of samples that were assigned to the model during runtime. The samples are collected in a buffer T^t which serves as a single training set. The update process first adds the new sample to T^t and then trains the linear model and its domain with T^t. Obviously, the usability of the algorithm would be quite limited if the size of the buffer is not. The computational costs in time and memory could easily become exhausting. The buffer could grow arbitrarily as long as the system keeps running and collecting new samples. As a result of, the re-training of the linear models and domains would become too time demanding. Beside such practical problems, a HLAM could not adapted to a dynamically changing target function. No sample of an input-output correspondence would be forgotten. Over time, the linear models would have to solve the impossible task to cope with data that contradicts itself. Hence the size of the buffer T^t is limited by another meta-parameter called *maxBufferSize*. The access to the buffer has to be implemented like a queue, so that the oldest sample will be discarded if a new sample is available and the buffer's maximal size is reached. How this can be done exactly is left to a programmer. In the pseudo code, it keeps hidden behind the procedure `RemoveOldestSample`. The parameter *maxBufferSize* has to be chosen by the user in accordance with the assumptions about the dynamics of the target function and the available memory and CPU power. The buffer should be small if the target function is quickly changing and the computational burden has to be kept low. In any case *maxBufferSize* must be larger or equal to the other meta-parameter *minSamples*.

So, the question remains which linear model should be updated by this process. The basic idea is to adapteth this model to the new sample which domain contains or is the nearest to it. Therefore the algorithm selects the model that has the highest value of $d_k(x)$ defined with (4). It is important to use the function $d_k(\cdot)$ instead of the proper activation function $a_k(\cdot)$ because these drop to minus infinity at the boundary of a domain. Hence those domains, that do not contain the new sample, have the same activation value. So, it can not be decided which domain is the nearest to the sample. Only domains that enclose the new sample, hence overlap each other, could be discriminated with their activation function value. But if only such domains would be taken into account for an update, domains could never expand, instead their size would shrink over time. The reason for that is grounded in the update process with the history buffer. The size of a domain is determined by the samples that are located at the boundary of the domain. But these will eventually be pushed out of the history buffer by samples that must lay inside of the domain to be accepted for an update. Hence the new samples from within the former domain will define the new but tightened boundary. In contrast to that, a domain can expand if samples are used for an update that are in the vicinity and not necessarily inside of it. This conception is implemented by the selection mechanism that updates this model that has the highest value of $d_k(x)$.

If the approximation performance of a HLAM is not satisfying for the new sample, the online algorithm improves the network by following two ideas. If a linear model already exists that should handle the new sample, this model will be updated with it. Additionally, the structure of the network may be changed by inserting a new linear model. The first idea is implemented in a straightforward manner taking advantage of the sharp boundaries of the SVD model: The model with the highest activation value $a_k(x)$ is checked if its domain contains the new sample, and updated in such a case.

The second idea needs more effort to be implemented. It relies essentially on the so called Neighborhood Graph G which was proposed in [7]. The graph G is given as a symmetric $\mathbb{N}^{M\times M}$ matrix and expresses which domains of a

TABLE II

RESULTS OF THE ABALONE BENCHMARK TEST.

Method	Number of Hidden Units	RMSE	
		Train Set	Test Set
HLAM offline	22	0.0440	0.0471
HALM online	20	0.0441	0.0477
RANEKF [9]	144	0.0655	0.0601
GAP-RBF [8]	18	0.0670	0.0613
MRAN [17]	33	0.0764	0.0669
RAN [12]	143	0.0838	0.0768
MRAN-OLS [11]	19	0.0965	0.0901

HLAM are adjacent to each other. The graph offers the advantage that a set of samples can be grouped w.r.t. already established domains of the HLAM. It clusters samples together that have the same domains as their nearest neighbors, hence are located in the same region of the domain space. Given *G*, the algorithm inserts new domains in those regions where enough samples could be collected. Therefore every new sample that is not approximated good enough is first stored in the extra buffer B^t. Then the neighborhood graph G is computed for these samples. The components of *G* represent the number of samples located in the neighborhood of these domains that correspond to the indices of the component. So, if there exists a component $G_{l,s}$ that is equal or larger than the threshold *minSamples*, a new linear model is created with a domain that is located in the vicinity of the *l*-th and *s*-th domain. Therefore these samples belonging to this local region are removed from the buffer B^t and used as a training set for the new linear model and its domain. Again, to simplify the pseudo code the definitions of the procedures `GetSamples` and `RemoveSamples` are omitted. They only have to manage the samples with an indexing system in correspondence to the domains.

One unsolved problem with this buffer is that in theory it may grow unboundedly. Samples will only be removed from the buffer if enough are available in the vicinity of two domains. Hence samples may never be used to establish a new local model, hence their information will be lost. The chances that the unused samples are in this sense badly distributed between domains grow with the number of local models. In the worst case the buffer would contain $\frac{1}{2}M(M-1)(minSample-1)$ samples since $\frac{1}{2}M(M-1)$ is the number of possible pairwise combinations of *M* domains.

In [7] we proposed a method to fuse domains of a HLAM in order to reduce its number of linear models. Although it was not designed for an online learning scenario, it is ready to be applied in it. The fusion algorithm can be started in regular intervals after an appropriate number of samples were received. Essentially, it tests if one linear model can be trained to approximate the unified training data of two adjacent domains. If this is the case, the domains will be fused and the new linear model will replace the two former ones. The modifications of the algorithm to the new SVD model are so straightforward that they can be omitted here.

IV. BENCHMARK EXPERIMENTS

In order to compare the new developments with other techniques three benchmark tests were performed.

TABLE III

RESULTS OF THE AUTO-MPG BENCHMARK TEST. FOR THE 50 TRIALS THE MEAN AND STANDARD DEVIATION OF THE RESULTS ARE GIVEN.

Method	Mean Number of Hidden Units	Mean RMSE	
		Train Set	Test Set
HLAM offline	$2.00 \pm .00$	$.0741 \pm .002$	$.0793 \pm .008$
HLAM online	$8.88 \pm .72$	$.0793 \pm .006$	$.0872 \pm .014$
MRAN [17]	$4.46 \pm .74$	$.1086 \pm .010$	$.1376 \pm .023$
RANEKF [9]	$5.14 \pm .90$	$.1088 \pm .012$	$.1387 \pm .029$
GAP-RBF[8]	$3.12 \pm .75$	$.1144 \pm .013$	$.1404 \pm .027$
MRAN-OLS [11]	$2.10 \pm .30$	$.1523 \pm .010$	$.1471 \pm .011$
RAN [12]	$4.44 \pm .84$	$.2923 \pm .081$	$.3080 \pm .092$

Mackey-Glass Data Set: In [10] a benchmark test with the chaotic Mackey-Glass differential equation was described. The experiment stems originally from [13] and defines the task to predict a value of the time series

$$x(t-1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \qquad (7)$$

with the parameters chosen to be: $a = 0.1$, $b = 0.2$, $\tau = 17$ and the initial condition as $x(0) = 1.2$. The function $f : \mathbb{R}^4 \to \mathbb{R}$ that has to be approximated is defined as

$$x(t+6) = f\Big(x(t), x(t-6), x(t-12), x(t-18)\Big). \qquad (8)$$

For a comparison of different machine learning techniques two separate sets of samples of f were generated. One set with $t = 124, \ldots, 1123$ serves as a training set for the machine learning technique, while the other one with $t = 1124, \ldots, 2213$ is used to calculate the RMSE as the final test error.

Abalone Data Set: The goal of this benchmark test is to estimate the age of an abalone given seven continuous and one discrete attributes. Instead of cutting the shell, staining it and counting the number of rings through a microscope the age should be predicted with attributes that are easier to obtain. Therefore, the sex, length, diameter, height and the weight of four different parts of an abalone are available as input values in the database [3] of 4177 samples. The target output value is discrete and ranges between 1 and 29 years.

As described in [11] the input and output values were normalized to the range $[0.1, 0.7]$. For the training 3000 samples were randomly selected from the whole data set. The RMSE was calculated for the remaining 1177 sample.

Auto-Mpg Data Set: In [15] the so called Auto-Mpg problem was stated to predict the city-cycle fuel consumption in miles per gallon in terms of three multivalued discrete and four continuous attributes. The data set that is available at [3] contains 398 samples of this continuous target function. As multivalued discrete input values the number of cylinder, the model year and the origin are given. The continuous attributes are the displacement, horsepower, weight and acceleration of the cars.

The benchmark test was repeated as described in [11]. Therefore the input and output values were normalized to the range $[0, 1]$. Since the number of samples is quite limited, a number of 50 trials were performed with different training and test sets. For each trial, 320 samples were randomly chosen for training, while the remaining 78 samples were

TABLE IV

VALUES OF META-PARAMETERS OF THE HLAMS USED FOR THE BENCHMARK TESTS.

	Meta-Parameters	Mackey-Glass	Abalone	Auto-Mpg
Offline	*minSamples*	2	100	150
	maxError	0.001	0.0001	0.0005
	σ	0.01	0.01	0.5
	ν	.0001	0.01	0.1
Online	*minSamples*	5	100	25
	maxBufferSize	20	150	50
	maxError	0.0001	0.0001	0.001
	σ	0.01	0.01	0.5
	ν	0.0001	0.01	0.1

used to compute the RMSE. The mean value of the trials' RMSE are listed as the final result of this benchmark test.

Results: Our approach with the off- and online learning algorithm was compared with the above described benchmark tests to the results reported in [10] and [11]. All results are listed in Table I, II and III. As kernel functions for the SVD model we used radial basis functions. The values of the meta-parameters for the learning algorithms are given in Table IV. They were selected by a process of repeated training and testing. Therefore, different values for *minSamples,* the error threshold *maxError* and *maxBufferSize* (only for the online algorithm) were chosen from an appropriate interval to train a HLAM[1]. The achieved RMSEs on the test set were compared and the best one is cited as the final result.

The tables show the superiority of the HLAM approach over the other methods w.r.t. the achieved RMSE in all benchmark tests. Both, the off- and the online learning algorithm performed very well. Especially in the Mackey-Glass benchmark test, the offline HLAM realizes a clearly better result than the alternatives. In every case our offline learning algorithm outperforms the proposed online version. So, the general expectation that an offline learning scenario is easier to handle with a machine learning technique is met. It can be pointed out that the new HLAM online learning algorithm exceeds the approximation quality of such offline algorithms like e.g. RANEKF or MRAN.

To illustrate the point that the flexibility of the SVD model helps to reduce the number of needed linear models, one HLAM was specially trained for the Mackey-Glass benchmark test (marked with a star in Table I). With the meta-parameter *minSamples* set to 20, the achieved RMSE degraded slightly (although still better than the competitors) but the number of models could be reduced by more than 50 %.

V. CONCLUSION

Along with new benchmark tests, we presented two new developments for our Hierarchical Network of Locally Arranged Models. A domain model that is based on a one-class support vector machine was proposed as an alternative to the former hyper-elliptical one. It features a more flexible domain boundary that helps to improve the approximation quality of a HLAM and to reduce the number of needed linear models. On the other hand, an online learning algorithm for the HLAM approach was delineated. It is designed to be used in an application where training data is only available as a continuous stream of samples. It allows to adapted a HLAM to a function that may change over time. Consistent with the basic HLAM idea, the online learning algorithm promotes specialization of the linear models to local regions of the domain space. The success of these two developments could be proven with three benchmark tests.

[1]Although in the Auto-Mpg benchmark test 50 HLAMs have to be trained for 50 different training and test sets the meta-parameters were only optimized once.

ACKNOWLEDGMENTS:

The work presented here was supported by the the European Union, grant COSPAL (IST-2003-004176). However, this paper does not necessarily represent the opinion of the European Community, and the European Community is not responsible for any use which may be made of its contents.

REFERENCES

[1] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2:302–309, 1991.

[2] K. B. Cho and B. H. Wang. RBF based adaptive fuzzy systems and their applications to system identification and prediction. *Fuzzy Sets and Systems*, 83:325–339, 1996.

[3] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer, 2001.

[5] F. Hoppe and G. Sommer. Local linear models for system control. In *Proceedings of Int. Conf. on Neural Information Processing (ICONIP)*, pages 171–176, 2005.

[6] F. Hoppe and G. Sommer. Ensemble learning for hierarchies of locally arranged models. In *Proceedings of IEEE World Concress on Computational Intelligence*, pages 10612–10619, 2006.

[7] F. Hoppe and G. Sommer. Fusion algorithm for locally arranged linear models. In *18th International Conference on Pattern Recognition (ICPR'06)*, pages III: 1208–1211, 2006.

[8] Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan. An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(6):2284–2292, 2004.

[9] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learningwith neural networks. *Neural Computation*, 5(6):954–975, 1993.

[10] Loo Chu Kiong, Mandava Rajeswari, and M. V. C. Rao. Extrapolation detection and novelty-based node insertion for sequential growing multi-experts network. *Neural Network World*, 2:151–176, 2003.

[11] Xiaoping Lai and Bin Li. An efficient learning algorithm generating small rbf neural networks. *Neural Network World*, 15:525–533, 2005.

[12] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[13] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[14] John C. Platt, Bernhard Schölkopf, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report MSR-TR-99-87, Microsoft Research (MSR), November 1999. An abridge version of this document will appear in *Neural Computation*.

[15] J. Ross Quinlan. Combining instance-based and model-based learning. In *ICML*, pages 236–243, 1993.

[16] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[17] Lu Yingwei, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation*, 9(2):461–478, 1997.