Ensemble Learning for Hierarchies of Locally Arranged Models

Florian Hoppe and Gerald Sommer
Department of Cognitive Systems, Institute of Computer Science and Applied Mathematics,
Christian Albrechts University, Kiel, Germany.

***Abstract*— We propose an ensemble technique to train multiple individual models for supervised learning tasks. The new method divides the input space into local regions which are modelled as a set of hyper-ellipsoids. For each local region an individual model is trained to approximate or classify data efficiently. The idea is to use locality in the input space as an useful constraint to realize diversity in an ensemble. The method automatically determines the size of the ensemble, realises an outlier detection mechanism and shows superiority over comparable methods in a benchmark test. Also, the method was extended to a hierarchical framework allowing a user to solve complex learning tasks by combining different sub-solutions and information sources.**

I. Introduction

Ensemble machine learning techniques solve a modelling task by using a specific training and combination scheme for an ensemble of individual models. Given a data set of input-output pairs, each individual model is trained to approximate a target function or to realise a classification. By combining these individual models, the approximation or classification performance of the full ensemble is hopefully better than the one of all singular models. Ensemble techniques can be regarded as meta machine learning methods as they typically employ common learning techniques (e.g. multi-layer perceptrons, decision trees, etc.) in their specific training and combination schemes. The crucial point of an ensemble training scheme is to realise diversity between the individual models since obviously a set of equal models could not be better than a single member of it. On the other hand, given a certain input, a good combination scheme has always to select the right subset of the trained models to combine their output as the ensemble's output.

The probably best known examples for ensemble techniques are *Bagging Predictors* [1], *AdaBoost* [2] and *Hierarchical Mixture of Experts* (HME) [3]. Especially Boosting techniques have in recent years gained lots of interest as they are grounded in the *Probably Approximately Correct* learning theory (cf. [4]) and were successfully applied to real world problems (for a detailed discussion cf. [5]).

Not that well known in the field of ensemble techniques are local (or also called piecewise) linear approximation (LLA) methods [6], [7], [8], [9]. Although they are typically not described as ensemble techniques, LLA methods rely on the same idea of combining multiple individual models to one more powerful approximator. In LLA methods, a set of linear models is used to approximate a non-linear function. Ensured by the training scheme, each of these linear models is only valid in a local region of the input space. The strategy of LLA is to achieve good global approximation by the right combination of good local approximators.

In this paper we want to propose the extension of our LLA method [9] to a full meta machine learning method for classification and approximation tasks. That is a method, which can employ any known machine learning techniques for the training of an individual model. Therefore, we compare in Sec. II the above mentioned ensemble techniques with common LLA methods. This will reveal similarities between these two research directions with three benifical effects: First, it grounds the straightforward extension of our learning scheme described in III. Second, it suggests that the locality constraint in LLA methods introduces fruitful diversity to an ensemble of models. Last but not least, the comparison with HME approach motivates our proposal in Sec. IV to build a hierarchically structured ensemble of our local models. In Sec. V, a benchmark test on the chaotic Mackey-Glass equation demonstrates the superiority of our learning scheme over other LLA methods, before a discussion of the presented ideas concludes this paper.

II. State of the Art Ensemble and LLA Techniques

All the bagging, boosting, HME and LLA approaches can be formalized as:

$$\hat{\mathbf{y}}(\mathbf{x}) = \sum_{k=1}^{N} g_k(\mathbf{x})\hat{\mathbf{y}}_k(\mathbf{x}), \tag{1}$$

where $\hat{\mathbf{y}}$ denotes the m-dimensional output of the ensemble for the n-dimensional input $\mathbf{x}$. The ensemble output $\hat{\mathbf{y}}$ is the sum of the outputs of the N individual models $\hat{\mathbf{y}}_k$, each weighted with the scalar valued functions $g_k(\mathbf{x})$.

Ensemble techniques do not define how the individual models $\hat{\mathbf{y}}_k$ are formalized. As meta machine learning techniques, they leave it to a system designer to choose a known learning method to train the models. Instead they can be differentiated by their specific function definitions for the weighting functions $g_k(\mathbf{x})$, by their method to determine the number N of used models and by their training schemes to produce diverse individual models. Latter has to go without a direct modification of the models' internal parameters since ensemble techniques should work with any kind of model. Instead diversity is introduced into the ensemble by indirect means. In the case of bagging and boosting, this is achieved with different methods of resampling a given set of training samples. In the HME approach, a special definition of the

weighting functions introduces a locality constraint which results in more competitive i.e. diverse models.

To lay the ground for a comparison between ensemble and LLA methods, in the next sections typical examples of these techniques are described.

A. Ensemble Techniques

A quite easy to implement ensemble technique is the *Bagging Predictor* [1] approach. Besides an appropriate machine learning technique (MLT) the system designer chooses first the number N of individual models constituting the ensemble. Then N subsets of training samples are compiled by uniform sampling with replacement from the originally given training set. For each subset a model is trained by means of the MLT. The outputs of the individual models are combined by setting $g_k(\mathbf{x}) = 1/N$, i.e. they are averaged. Note, that $g_k(\mathbf{x})$ is constant and not depending on the input $\mathbf{x}$.

In [1] it is argued that the benefit of bagging is variance reduction (cf. [10] for a discussion of the bias-variance trade off). As machine learning techniques are sensitive to different samples drawn from the same distribution, the trained individual models will generalize differently. Averaging will compensate such variance.

Boosting techniques can be exemplified with its first and famous *AdaBoost* [2] implementation (for a detailed overview about boosting see [5]). As in the bagging approach *AdaBoost* trains a number of different models using different samplings with replacement of the originally given training set. In contrast to bagging, the samples from the training set are not drawn with an uniform distribution and their output is not averaged but weighted with different g_k factors. Additionally, the number N of models is determined automatically. Therefore, *AdaBoost* runs in a loop in which each cycle adds a new model until a certain performance measurement is fulfilled. Each model is trained on a subset sampled with a non-uniform distribution. The sampling distribution depends on the performance of the model added in the last cycle. The distribution is changed so that the probability of a sample to be taken into the new training set rises with its approximation or classification error. The weighting factor g_k of the k-th model depends on its own performance on the training set: Those models with high error rates have small g_k values. As in the case of bagging the weighting functions are constant, not depending on the input $\mathbf{x}$ and normalized so that $\sum_k g_k(\mathbf{x}) = 1$ and $\forall k : 0 \leq g_k(\mathbf{x}) \leq 1$.

The essential strategy of all boosting algorithms is twofolded: During the iterative training more and more emphasis is laid on training samples which are hard to learn. On the other hand, models which show good performance will most prominently define the output of the whole ensemble. This matches well the successful strategy of Support Vector Machines (SVM). There, samples close to decision boundary which are typically hard to be correctly classified are constituting the learned model. In [5] this relation between Boosting and SVM techniques is also theoretically grounded.

The basic idea behind the *Hierarchical Mixture of Experts* (HME) approach [3] is that besides a set of expert networks (i.e. the ensemble of individual models) a so called gating network (i.e. the weighting functions $g_k(\mathbf{x})$) is trained that selects depending on the input $\mathbf{x}$ the most appropriate experts to compute the output $\hat{\mathbf{y}}$. So, in contrast to the Bagging and Boosting techniques the weighting functions are not constants but they are functions of the input pattern. In [3] linear models are used in combination with the *softmax* normalization [11] as weighting functions. Each of these weighting functions $g_k(\mathbf{x})$ divides the input space along a linear margin into two halves. All together, they realise a "soft" partitioning of the input space in regions where different expert networks are assigned to compute the ensemble's output $\hat{\mathbf{y}}$. The partitioning is called "soft" since the output $\hat{\mathbf{y}}$ is a weighted mixture of the experts' outputs $\hat{\mathbf{y}}_k$ which are also linear models. For a user specified number of N expert networks, an EM-Algorithm [12] approach was implemented in [3] to train the parameters of the expert networks and of the linear models for the weighting functions.

The authors of [3] also proposed to use HME networks them self as expert networks. Then a hierarchy of gating networks would compose a tree-like structure of expert networks. This has the effect that the "soft" partitioning induced by a higher-level gating network is subdivided into smaller parts by a gating network at a lower level.

The HME approach works as an ensemble technique as the implemented training scheme ensures that the individual models $\hat{\mathbf{y}}_k(\mathbf{x})$ are specialized according to the partitioning induced by the gating networks. One single model is not compelled to represent the target function in the whole input space but only in a part of it. This relaxation is only possible with non-constant weighting functions $g_k(\mathbf{x})$. Only if the weighting factor of an expert network changes for different input patterns, specialization on such input patterns can take place. Specialization means diversity between individual models, means typically more success of an ensemble of individual models.

B. Local Linear Approximation Methods

In contrast to the mentioned ensemble techniques, the LLA methods of [6], [8], [7], [9] were not proposed as meta machine learning but as specialized function approximation techniques. They were envisioned as methods for approximating a possible non-linear target function by the combination of linear models which are only valid in different but maybe overlapping regions of the input space. Such local regions are commonly called the domain of a linear model. Most often LLA methods were proposed for adaptive control tasks since sophisticated knowledge about linear systems from both control theory (important for stability proofs) and statistics (important for parameter estimation) is applicable. They fit nicely into the formalization of Eq. (1): The individual linear models[1] are defined as $\hat{\mathbf{y}}_k(\mathbf{x}) = \mathbf{C}_k\mathbf{x} + \mathbf{s}_k$ with the coefficient matrices $\mathbf{C}_k \in \mathbb{R}^m \times \mathbb{R}^n$ and the constant shifting vectors $\mathbf{s}_k \in \mathbb{R}^m$. The weighting functions $g_k(\mathbf{x})$

[1]For reason of simplicity, in this paper only linear models are discussed. The extension to polynoms of higher degrees is straightforward.

are modelling the domains of each linear model as functions of input $\mathbf{x}$: Depending on the distance of $\mathbf{x}$ to the position of the domain the weighting factors are larger or smaller and hence, will weight the output of the different linear models differently. Given a set of training samples $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}$, the different LLA training schemes try to minimize the approximation errors of the training samples which are located in the domain of a model. On the other hand, training samples which belong to another domain could have arbitrarily bad approximation results.

The common way (e.g. [6], [7]) to define a model's domain is to calculate the weighting factors according to the Gaussian function with a center vector $\mathbf{c}_k \in \mathbb{R}^n$ and a spread matrix $\mathbf{D}_k \in \mathbb{R}^n \times \mathbb{R}^n$:

$$g_k(\mathbf{x}) = \exp\Big(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x} - \mathbf{c}_k)\Big). \qquad (2)$$

The idea is that each linear model has as its domain a symmetrical receptive field centered at $\mathbf{c}_k$. Furthermore the weighting factors are typically normalized, so that a partition of unity of the input space is realized, i.e. that: $\sum_k g_k(\mathbf{x}) = 1$ and $\forall k : 0 \leq g_k(\mathbf{x}) \leq 1$.

It is part of the described strategies that such domains will overlap and hence, that the output of linear models will be mixed. That this strategy really improves the approximation result depends essentially on the right choice of overlap between the domains. The problem is that with weighting factors of the form of Eq. (2) (e.g. [6]), the overlap can only be controlled with the center $\mathbf{c}_k$ and the spread matrix $\mathbf{D}_k$. But due to the normalization, the shape of the domains will be distorted. As discussed in [13], the form of a radial basis function is no longer uniform, its maximum can be shifted from the center vector and the function value may not decrease monotonically with increasing distance to the center.

Given these architectures, the essential problem of LLA methods remains to find the right set of domains. Good approximation of the global function can be achieved only with a good partitioning of the input space into a set of regions where a linear model can efficiently approximate the target function. Therefore, the parameters of the weighting factors g_k must be optimized. The parameters of the linear models are typically determined by means of a weighted least square scheme, where the training samples are weighted by the factors g_k.

To decide how many linear models the ensemble should contain, the approaches commonly follow a strategy to incrementally insert and remove linear models. For example in [6], new models are included to the ensemble if for a given input $\mathbf{x}_j$ no weighting factor $g_k(\mathbf{x}_j)$ is larger than an user defined threshold. In this approach, linear models are also removed from the ensemble according to some similar threshold-based criterion. The methods differ in details where new models are inserted (i.e. what initial value is used for the center vector of a new model), but follow the same strategy to add models where the approximation error is high.

On the other hand, the approaches differ considerably in their way to determine the parameters of the weighting functions. A two-phase learning scheme is proposed in [7]: the center vectors are changed with Hebbian adaptation steps and by a gradient decent approach to minimize the least squares error function $J = \sum_{j=1}^{t} (\hat{\mathbf{y}}(\mathbf{x}_j) - \mathbf{y}_j)^2$ over the training set. In contrast to that, the authors of [6] prefer to minimize a so-called locally weighted error function which emphasizes that training samples only effect the domain's parameter that they belong to. As discussed in [3], [14] the latter suits better the overall strategy of mixture techniques since it improves local approximation by promoting competition between different models. The danger of too strong competition is that of overfitting: The locally best but globally worst solution would have as many models as training samples centered on the samples and having a very narrow spread matrix.

III. Ensemble of Locally Arranged Models

In [9], we presented a LLA method with a new domain model and learning algorithm. Its extension to a full meta machine learning technique with a more advanced domain model and a redundant model removal mechanism is proposed in the next section. The development is driven by the idea that locality in the input space provides an useful constraint to train diverse models and to combine them in an ensemble. Thereby, locality essentially means that data samples have small distances to each other according to some distance measurement (in our case the Euclidean distance).

A. Locally Arranged Models

In our approach, the output $\hat{\mathbf{y}}$ of the whole ensemble is the output of exactly one individual model. The model whose domain contains the input data $\mathbf{x}$ is selected to compute $\hat{\mathbf{y}}$. Therefore, a model's domain is given as a set of hyperellipsoids with sharp boundaries between the inside and the outside of the domain. One hyperellipsoid defines one part of a domain and is parameterized with a Gaussian-like distance function. The whole domain is the combination of such parts which are located adjacent to each other in the input space. Thereby, inspired by the neighborhood edges of a Dynamical Cell Structure (DCS) network [15], adjacency is measured as how many data samples are located between the centers of the hyperellipsoids.

So, in contrast to known LLA techniques, this scheme does not mix the output of individual models. The models are trained and used separately. With the sharp boundaries between domains the input space is divided into disjunctive regions associated to exactly one individual model. This realises a partition of unity which is easier to control than with the normalization used in the state of the art LLA approaches. Since more than one hyperellipsoids can be used to define a model's domain, more complex shaped domains can be modeled. Even gaps inside a domain can be expressed as the adjacency measurement does not depend on some mutual overlap between the hyperellipsoids. The neighborhood definition is purely data driven and impels

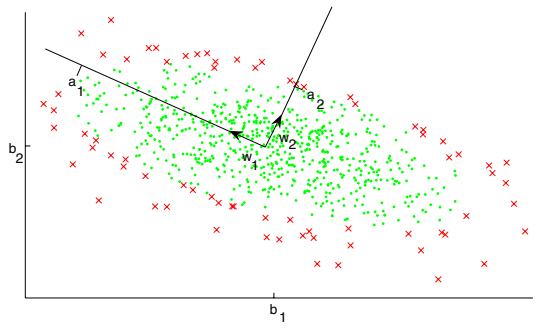

Fig. 1. Example of parameters of a model's 2D domain enclosing a set of data points. Inliers and outliers of the domain are shown as points and crosses, respectively.

no edge to exist between local parts as by Self-Organizing Maps [16].

As the base for our domain model, a hyperellipsoid is defined with the distance function $d(\cdot)$:

$$d(\mathbf{x};\mathbf{a},\mathbf{b},\mathbf{W}) = \exp\left[-\sum_{i=1}^{n}\left(\frac{1}{a_i}(\mathbf{x}-\mathbf{b})^T\mathbf{w}_i\right)^2\right]. \quad (3)$$

This parameterization of a Gaussian function has a clear interpretation: The n-dimensional hyperellipsoid is centered in the input space at position $\mathbf{b} \in \mathbb{R}^n$. The axes of such an hyperellipsoid are aligned along the coordinate axes $\mathbf{w}_i \in \mathbb{R}^n$ ($\mathbf{W} = (\mathbf{w}_1 \dots \mathbf{w}_n)$) and stretched with scalar valued factors $\mathbf{a} = (a_1, \cdots, a_n)^T$ with $a_i \geq 0$. One special case has to be treated: if there exists an i, so that $a_i = 0$ and $x_i \neq b_i$, then $d(\mathbf{x}) = 0$. See Fig. 1 for a illustrative 2D example. For purpose of convenience, the values of d are normalized by means of the exp-function to the interval $d(\mathbf{x}) \in [0, 1]$. The function has it's maximum at $\mathbf{b}$ and decreases along the hyperellipsoid's axes $\mathbf{w}_i$ to zero.

The activation $\alpha_k(\mathbf{x})$ of the k-th model is calculated as:

$$\alpha_k(\mathbf{x}) = \begin{cases} \max_l d(\mathbf{x};\mathbf{a}_l,\mathbf{b}_l,\mathbf{W}_l) : & d(\mathbf{x};\mathbf{a}_l,\mathbf{b}_l,\mathbf{W}_l) \geq \gamma_l \\ 0 : & else \end{cases}, \quad (4)$$

where $\{(\mathbf{a}_l, \mathbf{b}_l, \mathbf{W}_l, \gamma_l)\}$ is the set of parameters of the hyperellipsoids composing the whole domain. The scalar threshold parameter γ_l determines if $\mathbf{x}$ is enclosed by the l-th hyperellipsoid or not. The activation is defined so that $\alpha(\mathbf{x})$ equals the value of the distance function $d(\cdot)$ of the best matching hyperellipsoid. If $\mathbf{x}$ does not belong to any hyperellipsoid of the k-th model its activation is zero.

Finally, the weighting functions $g_k(\mathbf{x})$ can be defined as

$$g_k(\mathbf{x}) = \begin{cases} 1 & : \quad k = \arg_i \max \alpha_i(\mathbf{x}) \\ 0 & : \quad else \end{cases}, \quad (5)$$

which are all set to zero except the one with the maximal activation. One special case arises when all $\alpha_i(\mathbf{x})$ equal zero, which means that no hyperellipsoid contains $\mathbf{x}$. In our architecture such data $\mathbf{x}$ is treated as an outlier and automatically rejected as an invalid input. Essentially, the threshold parameters γ_l establishes this automatic outlier detection. This is not possible with the approaches [6], [7] since weighting factors of the form of Eq. (2) define unlimited domains.

On the other hand, if an outlier detection is not appropriate for the application at hand, one still can assign a model to $\mathbf{x}$ according to some rule. In our experiments, we are making a nearest neighbor decision which chooses the model with the smallest Euclidian distance between $\mathbf{x}$ and the center of one of its hyperellipsoids. This matches well the idea of associating models to local regions of the input space and experiments showed its effectiveness.

B. Ensemble Learning Algorithm

As the new model with its sharply bounded domains already emphasizes local validity of an individual model, the learning algorithm stresses this as well. The learning algorithm is a recursive scheme in which the training set is split into subsets belonging to local regions. To each subset an individual model is assigned to approximate the target function (to classify samples, respectively) in the corresponding region. Then the models are trained with the samples from the subsets by means of the MLT the system designer has chosen. The parameters of the model's domain are determined with a method described further below. This process of splitting and training is repeated as long as the performance of each individual model is not sufficient good and the subsets contain enough training samples. If a set of samples is split up, the model assigned to it will be removed from the ensemble. It will be replaced by other models whose domains will occupy the domain of the removed one.

After the recursive splitting and training process has stopped, the domains of all established models will be composed out of one hyperellipsoid. Then a domain fusion algorithm which removes redundant models will be started. It computes a neighborhood graph expressing which domains are adjacent to each other. Then it evaluates if the training samples from adjacent domains can well enough be learned with only one model. If that is the case, the two former domains will be joined and the two original models will be replaced with a new one trained on the joined domain. This process of computing a neighborhood graph and evaluating adjacent models will be repeated as long as domains can be fused. Noticeable about the other redundant model removal mechanisms of [6] and [7] is that with removing a linear model, information about already determined domains is lost since their domain model does not allow multiple center vectors for one single domain. In contrast, our fusion algorithm keeps the already gained information by combining domains to more complex ones.

In detail (see Algorithm 1 and 2): The learning algorithm starts with one individual model having the whole input space as its domain. Using all training samples within its domain the model's parameters Θ are trained with the chosen machine learning technique. Then the performance of the trained model has to be estimated in order to decide if the

Algorithm 1: Pseudo code for ensemble training with a generic machine learning technique (MLT).

Function `TrainEnsemble`
Input : $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}$
Output: $M = \{(\Theta_k, T_k, \mathbf{a}_k, \mathbf{b}_k, \mathbf{W}_k, \gamma_k)\}_{k\in[1,N]}$
begin
  $m \leftarrow$`TrainModel(`T`)`
  $M \leftarrow$`OptimizeModel(`m`)`
  $M \leftarrow$`RemoveRedundantModels(`M`)`
end

Function `TrainModel`
Input : $T^{in} = \{(\mathbf{x}_j, \mathbf{y}_j)\}$
Output: $m = (\Theta, T, \mathbf{a}, \mathbf{b}, \mathbf{W}, \gamma)$
begin
  $\Theta \leftarrow$ MLT(T^{in})
  $T \leftarrow T^{in}$
  $\mathbf{b} \leftarrow \frac{1}{|T|}\sum_j \mathbf{x}_j$
  $\mathbf{w}_i \leftarrow i$-th principle component of set $\{\mathbf{x}_j\}$
  $a_i \leftarrow \max_j \|(\mathbf{x}_j - \mathbf{b})^T \mathbf{w}_i\|$
  $\gamma \leftarrow \min_j d(\mathbf{x}_j; \mathbf{a}, \mathbf{b}, \mathbf{W})$
end

Function `OptimizeModel`
Input : $m = (\Theta, T, \mathbf{a}, \mathbf{b}, \mathbf{W}, \gamma)$
Output: $M = \{(\Theta_k, T_k, \mathbf{a}_k, \mathbf{b}_k, \mathbf{W}_k, \gamma_k)\}_{k\in[1,N]}$
begin
  PerformanceIsNotOK$\leftarrow$test performace of Θ
  if *PerformanceIsNotOK* $\wedge$ $|T|$ >*minSamples* **then**
    $\{T_l\} \leftarrow$`SplitTrainingSet(`T, Θ`)`
    forall *l* **do**
      $m_l \leftarrow$`TrainModel(`T_l`)`
      $M_l \leftarrow$`OptimizeModel(`m_l`)`
    $M \leftarrow \bigcup_l M_l$
  else
    $M \leftarrow \{m\}$
end

Function `SplitTrainingSet`
Input : $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}, \Theta$
Output: $R = \{T_l\}_{l\in[1,L]}$
begin
  forall *j* **do**
    $Err_j \leftarrow$approximation or classification error of Θ for $\mathbf{x}_j$
  $\{c_l\} \leftarrow$k-means to minimize $\sum_{l=1}^{L}\sum_{\mathbf{x}_j\in S_l} \|\mathbf{x}_j - \mu_l\|$ with $\mu_l = \sum_{\mathbf{x}_i\in S_l} \frac{Err_i}{\sum_j Err_j}\mathbf{x}_i$ and $S_l = \{\mathbf{x}_j \in T | l = \arg_i \min \|\mathbf{x}_j - \mu_i\|\}$
  forall *l* **do**
    $T_l \leftarrow \{\mathbf{x}_j \in T | l = \arg_i \min \|\mathbf{x}_j - \mathbf{c}_i\|\}$
end

model should be added to the ensemble or not. Depending on the task at hand the mean squared approximation error (MSE) or the classification rate on the training samples in a domain could be computed and compared to a user specified threshold. Since both criteria are based on the training data, they do not really express the generalization performance of the model but proved to be useful in our experiments. Depending on the used MLT, a criterion based on e.g. cross-validation might be more suitable. If the model's performance is not acceptable and if the number of samples exceeds a user chosen threshold $minSamples$, then the trained model will be rejected and replaced by other differently trained ones. Therefore the training set is split up with a special clustering method described below and new models are assigned to the generated subsets. For each new model the same training and evaluating process will be repeated until optimization is completed as no more model can be added.

Then the domains of the models will be trained. The basic idea therefore is to use *principle component analysis* (PCA) on a model's training set to determine its parameters $\mathbf{a}_l, \mathbf{b}_l, \mathbf{W}_l$ and γ_l. The eigenvectors of the covariance matrix of the training set's input samples are used as the axes $\mathbf{W}_l$ of the hyperellipsoid. As the factors a_i should express the variance along the i-th axis of the hyperellipsoid, they are set to the maximal projection of the input samples onto the i-th eigenvector. The position $\mathbf{b}_l$ is chosen to be the mean of the input samples. The boundary between the inside and the outside of a hyperellipsoid is drawn by setting the threshold γ_l to the minimal value of d for all the training samples.

A distinctive feature of the algorithm is how the training set of an individual model Θ is split up in order to optimize performance. A method was implemented which clusters the input samples $\mathbf{x}_j$ of a training set $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}$ w.r.t. a model's performance value Err_j for each $\mathbf{x}_j$. In case of a function approximation task Err_j equals the approximation error: $Err_j = \|\hat{\mathbf{y}}(\mathbf{x}_j; \Theta) - \mathbf{y}_j\|$. For a classification task Err_j is set to a user specified constant $goodClass \in [0, 0.5]$ or to $Err_j = 1 - goodClass$ if the predicted class label $\hat{\mathbf{y}}(\mathbf{x}_j; \Theta)$ equals the true class label $\mathbf{y}_j$ or not, respectively. The clustering method minimizes the function

$$\sum_{l=1}^{L}\sum_{\mathbf{x}\in S_l} \|\mathbf{x} - \mu_l\| \text{ with}$$

$$S_l = \{\mathbf{x}_j \in T | l = \arg_i \min \|\mathbf{x}_j - \mu_i\|\} \text{ and}$$

$$\mu_l = \sum_{\mathbf{x}_i\in S_l} \frac{Err_i}{\sum_j Err_j}\mathbf{x}_i$$

where L is the number of new clusters to be generated[2]. This is essentially the k-means algorithm [10, in Sec. 5.9.1] where the center vectors μ_l are set to the mean of a subset S_l weighted with Err_j. The normal k-means algorithm would generate clusters only driven by the distribution of the samples in input space. Whereas the weighted version ensures

[2]Although L is a free parameter of the algorithm, we only used $L = 2$.

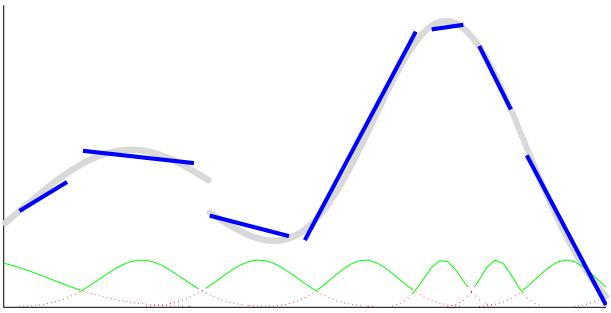

Fig. 2. Nonlinear function with discontinuity and its linear approximation. The domains of models are the bell shaped curves. The outside of a domain is plotted as a dotted line. Note, more linear models are used where the curvature of the target function is high.

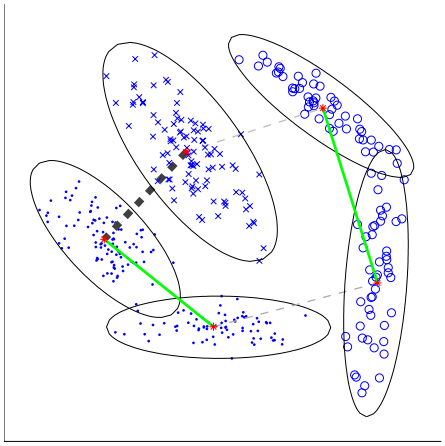

Fig. 3. 2D input sample for three linear models (dots, crosses and circles). Hyperellipsoids connected with a line belong to a fused domain. Dashed lines visualize the neighborhood graph between the three domains. Thicker lines express that more samples are laying between adjacent domains.

that the generated subsets are centered in regions where the performance of model Θ was bad. Like the *AdaBoost* algorithm this mechanism emphasizes badly learned training samples when a new model is trained with a different sampling of the original training set. Fig. 2 illustrates this with an ensemble of linear models for an one dimensional synthetical function approximation task. The figure shows that more linear models are trained where the target function is highly non-linear since there linear models yield higher approximation error. In case of a classification task, the user can adjust how much weight should be put onto badly learned training samples by the factor $goodClass$.

After the individual models and their domains are established, the domain fusion algorithm will be started (see Algorithm 2). The algorithm repeatedly computes the neighborhood graph between two domains and fuses them if a new model could successfully be trained on the data of both of them. The graph $\mathbf{G}$ is a symmetric $\mathbb{N}^N \times \mathbb{N}^N$ matrix, in which $\mathbf{G}_{k,l}$ and $\mathbf{G}_{l,k}$ equal the number of input samples $\mathbf{x}_j$ located directly between the k-th and the l-th domain. Therefore it is counted how many samples have a hyperellipsoid center from two different domains as their next and second next neighbor w.r.t. the Euclidean distance (see Fig. 3 for an example).

This graph matrix captures well which domains are adjacent to each other and where the most samples are located between two domains. Latter is used to decide which two individual models should be tested to be fused. So, with $(id_1, id_2) = \arg_{i,j} \max \mathbf{G}_{i,j}$ the training sets from domain id_1 and id_2 are united and a new model is trained by means of the machine learning technique. If the model's performance is acceptable, the former two models will be replaced by the new one and its new domain will be composed out of all hyperellipsoids of the two former domains. Then the repetition of graph computing and fusion tests will be started again. If the model's performance was not acceptable, the algorithm will be stopped and the ensemble is then completely trained.

IV. Locally Arranged Ensembles in a Hierarchical Framework

The *Hierarchical Mixture of Expert* approach lays the ground for a powerful extention of our ensemble method: Gating networks on different levels of a hierarchy of locally arranged models can easily be constructed with the above proposed weighting factors. Suppose a number of our ensembles are trained to solve different machine learning tasks. The difference of the tasks is expressed by different training sets with samples which occupy the same input and output spaces. Such ensembles can be combined to a two level hierarchy by defining domains for each ensemble with the same means as for individual models. Using the training set of each ensemble, the domain parameters $\mathbf{a}_k, \mathbf{b}_k, \mathbf{W}_k$ and γ_k of the weighting factors $g_k(\mathbf{x})$ for the 2^{nd} level gating net can be determined as described in Sec. III-B. Applying this scheme repeatedly, a system designer can combine different ensembles by build up such hierarchies with as many levels as needed. Of course even different machine learning techniques can be mixed: E.g. in one ensemble RBF networks and in the other SVMs could be applied. At different gating networks, also different input spaces can be used, so that various information sources can be employed.

For example, a hierarchical classification scheme for object categorization could be build up. On the lowest level specialized ensembles could be trained, one maybe to separate oranges from apples the other to distinguish cars from houses. Then on a second level, these ensembles could be combined by a gating network which learns to tell the difference between a natural and an artifical made object. The hierarchy could be extended arbitrarily in this way. One would get a powerful classification mechanism where, driven by the subcategories of a given object, the gating networks would select first the right subtrees of the ensemble and

Algorithm 2: Pseudo code for the domain fusion algorithm with a function L2K which maps the index of a center to the index of its domain.

Function RemoveRedundantModels
Input : $M^{in} = \{m_k = (\Theta_k, T_k, \mathbf{a}_k, \mathbf{b}_k, \mathbf{W}_k, \gamma_k)\}$
Output: $M^{out} = \{(\Theta_k, T_k, \mathbf{a}_k, \mathbf{b}_k, \mathbf{W}_k, \gamma_k)\}$
begin
  $G \leftarrow$GetNeighborhoodGraph $(\bigcup_k T_k, \bigcup_k \mathbf{b}_k)$
  $(id_1, id_2) \leftarrow \arg_{i,j} \max G_{i,j}$
  $\Theta \leftarrow$TrainModel $(T_{id_1} \cup T_{id_2})$
  PerformanceIsNotOK$\leftarrow$test performace of Θ
  if *PerformanceIsNotOK* **then**
    $fusedModel \leftarrow \Theta$ with $T = T_{id_1} \cup T_{id_2}$ and combined domains from m_{id_1} and m_{id_2}
    $M \leftarrow (M^{in} \setminus \{m_{id_1}, m_{id_2}\}) \cup \{fusedModel\}$
    $M^{out} \leftarrow$RemoveRedundantModels (M)
  else
    $M^{out} \leftarrow M^{in}$
  end
end

Function GetNeighborhoodGraph
Input : $T = \{(\mathbf{x}_j, \mathbf{y}_j,)\}, \{\mathbf{b}_l\}$
Output: $\mathbf{G} \in \mathbb{N}^N \times \mathbb{N}^N$
begin
  $\mathbf{G} \leftarrow 0$
  forall *j* **do**
    $k_1 \leftarrow$ L2K$(\arg\min_l \|\mathbf{x}_j - \mathbf{b}_l\|)$
    $k_2 \leftarrow$ L2K$(\arg\min_{\{l|\text{L2K}(l) \neq k_1\}} \|\mathbf{x}_j - \mathbf{b}_l\|)$
    $\mathbf{G}_{k_1,k_2} \leftarrow \mathbf{G}_{k_1,k_2} + 1$
    $\mathbf{G}_{k_2,k_1} \leftarrow \mathbf{G}_{k_2,k_1} + 1$
  end
end

eventually the right individual model to perform the classification. The proposed hierarchical framework helps to design such a complex classification scheme. The system designer can decide which information source is most appropriate to decide about specific subcategories. He can compile different training sets for the different classifiers at the lowest level in order to ensure correct classification. This would not be possible with a monolithic machine learning technique such as a Multi-Layer Perceptron trained to do the whole task at once.

This gating on different levels with the proposed domain model will work whenever at least one input dimension contains a constant-mean signal. The signal indicates which subtree of the hierarchy should compute the output of the whole ensemble. Such input dimensions would code the context information of data samples which is constant for a single subtree. The mean value and the noise level of this signal will automatically be captured in the domain parameters $\mathbf{b}$ and $\mathbf{a}$, respectively.

TABLE I
RESULTS OF THE BENCHMARK TEST WITH THE CHAOTIC MACKEY-GLASS EQUATION.

Method	#hidden units	RMSE
LALM with fusion	77	0.0017
LALM without fusion	96	0.0018
GMN [7]	7	0.0091
RBF-AFS [19]	21	0.0128
OLS [18]	132	0.0163

V. MACKEY-GLASS BENCHMARK EXPERIMENT

To compare our approach with state of the art *Local Linear Approximation* algorithms an experiment described in [7] with the chaotic Mackey-Glass differential equation was repeated. The experiment stems originally from [17] and defines the task to predict a value of the time series

$$x(t-1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \qquad (6)$$

with the parameters $a = 0.1$, $b = 0.2$, $\tau = 17$ and $x(0) = 1.2$. The function to be approximated is defined as $x(t+6) = f\Big(x(t), x(t-6), x(t-12), x(t-18)\Big)$.

The training and test set contains 1000 samples of f with $124 \leq t \leq 1123$ and $1124 \leq t \leq 2213$, respectively. Tab. I shows the achieved results of four other methods[3]: the Growing Multi Expert [7], the Orthogonal Least Squares [18], the Radial Basis Functions based on the Adaptive Fuzzy System [19] and two results with our ensemble method (called Locally Arranged Linear Models, LALM). Our ensembles had as individual models linear models which were trained with a least squared method. During the learning phase their performance is estimated by the MSE criterion described in Sec. III-B. One of our results was achieved with, the other without the domain fusion algorithm. We implemented our algorithm in MatLab (R14) running on a Linux PC with a 3000 MHz CPU (1 GB RAM). The training of a ensemble for this benchmark test took always less than 90 sec.

Our methods showed superiority w.r.t. the achieved RMSE. Note, that the domain fusion algorithm reduces the number of linear models by approx. 20% and the fused ensemble even achieves slightly better performance. Observable is that our solutions still need more linear models than the local linear method of [7].

VI. CONCLUSION

We proposed an ensemble machine learning technique to train and combine multiple individual models to solve a classification or function approximation task. The new method defines how a given training set is divided into subsets each used to train one individual model with a generic machine learning technique. The division into subsets represents a division of the input space into local regions. Such regions are called domains and modelled as a set of hyper-ellipsoids. The learning method guarantees that the training data in each domain is efficiently approximated or

[3]Besides our own, all results are quoted from [7].

classified by one individual model. Given a certain input, the model whose domain contains the input pattern is selected to compute the output of the whole ensemble.

The new method with its specific domain model and learning algorithm is an extension of a previously proposed Local Linear Approximation method to general meta machine learning technique. The system designer has only to chose a MLT for the actual training of individual models and a performance criterion which decides if such an individual model is trained well enough. The described learning algorithm then automatically determines how many models will constitute the ensemble and which input samples are handled by which individual model. The underlying idea is that locality in the input space provides an useful constraint to train diverse models and to combine them in an ensemble. Thereby, locality essentially means that data samples are similar to each other according to some distance measure (in our case the Euclidean distance). The effect of the locality constraint is that the individual models are trained in a competitive manner against each other. The result is that the ensemble is constituted of diverse and specialized models because these are only trained with samples from their sharply bounded domains. Specialization of single models is also the result of the proposed weighted clustering method as it emphasizes training samples which are hard to learn. In contrast to known techniques, our domain model also provides an automatic outlier detection which helps to decide if data can or can not be handled with a trained ensemble.

We also transfered the concept of a multi-level hierarchical framework into our ensemble method for locally arranged models. By using the same domain model of an individual model for a whole subtree of networks, we defined a generic way to combine trained ensembles to more complex ones. A system designer can thereby extend the competence of an already trained ensemble by adding new levels and more specialized models. He can even integrate different types of machine learning techniques into the hierarchy. This should help him to solve a complicated information processing task by distributing different signal sources over a hierarchy of gating networks to specialized models.

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] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[2] Freund and Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *JCSS: Journal of Computer and System Sciences*, vol. 55, 1997.

[3] M. I. Jordan and R. A. Jacobs, "Hierarchial mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, pp. 181–214, 1994.

[4] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[5] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," in *Machine Learning Summer School*, pp. 118–183, 2002.

[6] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, no. 8, pp. 2047–2084, 1998.

[7] L. C. Kiong, M. Rajeswari, and M. V. C. Rao, "Extrapolation detection and novelty-based node insertion for sequential growing multi-experts network," *Neural Network World*, vol. 2, pp. 151–176, 2003.

[8] M. Tagscherer, L. Kindermann, A. Lewandowski, and P. Protzel, "Overcome neural limitations for real world applications by providing confidence values for network prediction," in *Proceedings of Int. Conf. on Neural Information Processing (ICONIP)*, pp. 520–525, 1999.

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

[10] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995.

[11] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neuro-computing: Algorithms, Architectures and Applications* (F. Fogelman Soulié and J. Hérault, eds.), (Berlin), pp. 227–236, Springer, 1990.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm.," *J. Royal Stat. Soc. B*, vol. 39, no. 1, pp. 1–38, 1977.

[13] R. Shorten and R. Murray-Smith, "Side-effects of normalising basis functions in local model networks," in *Multiple Model Approaches to Modelling and Control* (R. Murray-Smith and T. A. Johansen, eds.), ch. 8, pp. 211–228, London: Taylor & Francis, 1997.

[14] J. Nakanishi, J. A. Farrell, and S. Schaal, "A locally weighted learning composite adaptive controller with structure adaptation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 882–889, 2002.

[15] J. Bruske and G. Sommer, "Dynamic cell structure learns perfectly topology preserving map," *Neural Computation*, vol. 7, no. 4, pp. 845–865, 1995.

[16] T. Kohonen, *Self-Organizing Maps*, vol. 30. Berlin, Heidelberg, New York: Springer, 1995, 1997, 2001.

[17] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[18] 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*, vol. 2, pp. 302–309, 1991.

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