Fusion Algorithm for Locally Arranged Linear Models

Florian Hoppe and Gerald Sommer
Cognitive Systems Group, Christian-Albrechts-University of Kiel, Germany.

Abstract

As an extension to a recently proposed local linear approximation method we present an algorithm that generates more compact solutions for supervised-learning problems. Given a network of linear models each trained to approximate the target function in a local region of the input space, the algorithm reduces the number of the models significantly without diminishing the accuracy of the approximation. It fuses linear models by combining their local regions of validity to more complex, non-symmetrically shaped ones. A neighborhood graph introducing edges in a purely data-driven manner between adjacent linear models is used to determine which models should be fused. The also extended model for a region of validity allows to detect automatically data which is novel to a trained network and should be regarded as an outlier. The effectiveness of the proposed methods is shown with a benchmark test achieving a five times smaller RMSE than the best competitors.

1 Introduction

Local linear approximation (LLA) methods were proposed as especially well suited machine learning techniques for system modelling and control problems [4, 5, 6, 8]. The common idea behind these learning techniques is to approximate a non-linear function by employing a number of linear models which are only valid for local regions of the input space. The major strategy is to achieve good global approximation by the combination of good local approximators. For control tasks this idea is very fruitful since using linear models as local approximators makes sophisticated knowledge from control theory (important for stability proofs) and statistics (important for parameter estimation) applicable.

In [4] we proposed a new approach to LLA. The key feature of our network model is that a region of validity of a linear model – called the domain of a model – has a sharp boundary to other domains. Since there is no overlap between domains there is no ambiguity which linear model should compute the output for a given input. Our approach stresses the general idea of LLA that input data can be grouped to clusters and then assigned to a linear model. As an extension to that we now present a new fusion algorithm which reduces significantly the number of linear models in a trained network by combining single domains to more complex ones. Its core idea is based on the neighborhood graph of the Dynamical Cell Structure network [1] which introduces edges in a purely data-driven manner between adjacent clusters.

2 State of the Art Techniques

Network models: Local linear approximators can be formalized as:

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

where $\hat{\mathbf{y}}$ denotes the m-dimensional output of the network for the given n-dimensional input $\mathbf{x}$. The N coefficient matrices $\mathbf{C}_k$ and the constant shifting vectors $\mathbf{s}_k$ are the parameters of the linear models. The scalar valued weighting factors $g_k(\mathbf{x})$ define the domains of the models $1 \leq k \leq N$. These are 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 $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}$ and a set of defined domains, the linear models are trained with linear regression methods. Goal is to minimize the approximation errors of the training samples which are located in the domain of a model.

The common way (e.g. [6, 5]) to define a model's domain is to calculate the weighting factors according to the Gaussian function with a center vector $\mathbf{c}_k \in \mathbb{R}^n$:

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

The idea is that each linear model has as its domain a symmetrical receptive field centered in $\mathbf{c}_k$. Furthermore the weighting factors are typically normalized, so that a partition of unity of the input space is realized.

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

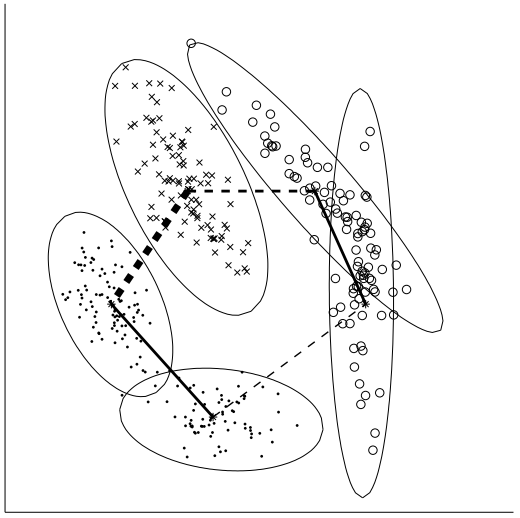

Figure 1. 2D input sample for three linear models (dots, crosses and circles). Hyperellipsoids connected with a line belong to a fused domain. Dashed lines visualize the neighborhood graph between the three domains. Thicker lines express that more samples are laying between adjacent domains.

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 [7], 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.

Learning Algorithms: Given these networks, the essential problem of local linear techniques remains to find the right set of domains. Good approximation of the global function can be achieved only with a good partition 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 network should contain, the approaches commonly follow a strategy to incrementally insert and remove linear models. Inserting and removing is performed according to some criterions based on some user-specified thresholds. Noticeable about [6] and [5] 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 (c.f. Sec. 3) keeps the already gained information by combining domains to more complex ones. In common approaches the parameters of the weighting factors are trained with a gradient decent method to minimize some special error function.

3 Locally Arranged Linear Models

Network model: In our approach, the output $\hat{\mathbf{y}}$ of the whole network will be the output of exactly one linear 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 a 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 [1], adjacency is measured as how many data samples are located between the centers of the hyperellipsoids.

So, in contrast to known techniques, this scheme does not mix the output of linear models. Linear models are trained and used separately. With the sharp boundaries between domains the input space is divided into disjunctive regions associated to exactly one linear model. This realises a partition of unity which is easier to control than with the normalization used in the state of the art 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 no edge to exist between local parts as by Self-Organizing Maps.

As the base for our domain model, a hyperellipsoid is defined as 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 it exists an i, so that $a_i = 0$ and $x_i \neq b_i$, then $d(\mathbf{x}) = 0$. 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 to zero.

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

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

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 will be zero.

Finally, the weighting factors g_k can be defined as

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

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 model such data $\mathbf{x}$ is treated as an outlier and automatically rejected as a valid input. Essentially, the threshold parameters γ_l establishes this automatic outlier detection. This is not possible with the approaches [6, 5] since weighting factors of 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 an expert to $\mathbf{x}$ according to some rule. In our experiments, we are making a nearest neighbor decision which chooses the linear model with the smallest Euclidian distance between $\mathbf{x}$ and the center of the hyperellipsoid $\mathbf{b}^k$. This matches well the idea of LLA associating linear models to local regions.

Learning Algorithm: The learning algorithm is a recursive scheme in which the training set is split into subsets belonging to local regions. To each subset an single linear model is assigned to approximate the target function in the corresponding region. Then the linear model and its domain are trained with the samples from the subsets. These processes of splitting and training are repeated as long as the approximation error of each linear model is not sufficient small 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 network. It will be replaced by other models whose domains will occupy the domain of the former one. After the recursive splitting and training process stopped, the domains of all established linear models will be composed out of one hyperellipsoid. Then the new domain fusion algorithm 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 be well enough approximated with only one linear model. If that is the case, the two former domains will be joined and the two linear models will be replaced with a new one trained on the joined domain. This process of computing a neighborhood graph and evaluating adjacent domains will be repeated as long as models can be fused.

In detail: The learning algorithm starts with one linear model having the whole input space as its domain.

Function `GetNeighborhoodGraph`
Input : $T = \{(\mathbf{x}_j, \mathbf{y}_j,)\}, \{\mathbf{b}_l\}$,`L2K`
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`$(\arg\min_{\{l|\texttt{L2K}(l)\neq k_1\}} \|\mathbf{x}_j - \mathbf{b}_l\|)$
    $\mathbf{G}_{k_1,k_2} \leftarrow \mathbf{G}_{k_1,k_2} + 1$
    $\mathbf{G}_{k_2,k_1} \leftarrow \mathbf{G}_{k_2,k_1} + 1$
  end
end

Algorithm 1. Pseudo code for computing the neighborhood graph given a training set, a set of hyperellipsoid centers and a function `L2K` which maps the index of a center to index of its domain.

The model's parameter $\mathbf{C}_k$ and $\mathbf{s}_k$ are trained with a least squares method using all training samples within its domain. The squared approximation error for each training sample are computed. If their mean is larger than a user specified threshold $maxError$ and if the number of samples exceeds another user chosen threshold then new added models will replace the currently optimized one. Therefore the training set is split up with a k-means-like clustering method (s. [4] for details) 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 complete. Then the domains of the models will be trained. The basic idea therefore is to use *principle component analysis* 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.

After the linear models and their domains are established, the new domain fusion algorithm will be started. The algorithm repeatedly computes the neighborhood graph between domains and fuses two appropriate linear models. 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. Therfore 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. 1 for an example and Alg. 1 for pseudo code).

Table 1. 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 [5]	7	0.0091
RBF-AFS [3]	21	0.0128
OLS [2]	132	0.0163

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 linear models should be tested to be fused. So, with $(k,l) = \arg\max_{i,j} \mathbf{G}_{i,j}$ the training set from domain l and k are united and a new linear model is trained with by means of a least squares method. The training was successful if the mean approximation error is smaller than the threshold $maxError$. If so, 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. Otherwise the algorithm will end.

4 Mackey-Glass Benchmark Experiment

For performance comparison an experiment described in [5] with the chaotic Mackey-Glass equation was repeated. 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 network was trained on 1000 samples of f with $124 \leq t \leq 1123$ and tested with $1124 \leq t \leq 2213$. Tab. 1 shows the achieved results of four other methods: the Growing Multi Expert [5], the Orthogonal Least Squares [2], the Radial Basis Functions based on the Adaptive Fuzzy System [3] and two results with our network. One after the first learning phase is completed and one after the new fusion algorithm was applied. We implemented our algorithm in MatLab (R14) running on a Linux PC with a 3000 MHz CPU (1 GB RAM). The training of a network for this benchmark test took always less than 90 sec.

Our methods showed superiority w.r.t. the achieved RMSE. Note that the new domain fusion algorithm reduces the number of linear models by approx. 20% and the fused network even achieves a slightly better performance. Observable is that our solutions still need more linear models than the local linear method of [5].

5 Conclusion

We proposed a new domain fusion algorithm for our local linear approximation approach [4]. It is based on a purely data-driven adjacency measurement employing the neighborhood graph of a DCS network. The adjacency definition stresses the idea of locality which is crucial for any LLA approach. Such a graph determines which domains are tested to be combined and assigned to one linear model. The domain model can therefore be a combination of more than one hyperellipsoid to one domain. In contrast to other approaches, this allows to assign a linear model to a data cluster with a complex, non-symmetrical shape. In practice, the fusion algorithm reduced the number of linear models significantly without diminishing the accurateness of the approximation. Another unique feature of the proposed domain model is that it provides an automatic outlier detection which helps to decide if data can be approximated with a trained network.

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] J. Bruske and G. Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845–865, 1995.

[2] 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.

[3] 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.

[4] 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.

[5] 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*, 2:151–176, 2003.

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

[7] R. Shorten and R. Murray-Smith. Side-effects of normalising basis functions in local model networks. In R. Murray-Smith and T. A. Johansen, editors, *Multiple Model Approaches to Modelling and Control*, chapter 8, pages 211–228. London: Taylor & Francis, 1997.

[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)*, pages 520–525, 1999.