Local Linear Models for System Control

Florian Hoppe, Gerald Sommer

Computer Science Institute, Christian Albrechts University of Kiel, Germany.

Abstract— **For function approximation in the context of system control we present a new learning algorithm for a network consisting of local linear models. The algorithm finds a partition of the input space in disjunct regions in which the target function can be "efficiently" approximated by a single linear model: in regions where the function has high curvature more models are positioned, while in regions where the target function is almost linear less models are trained. Due to the composition of linear models the network has a clear interpretation which makes it amenable for stability proofs when used as an adaptive controller. With the proposed algorithm the network is easy to train and tune. Their performance is demonstrated on the well-known Mackey-Glass benchmark test as well as with a robot control task.**

I. Introduction

Correct system modelling is the key for successful controlling. Modelling a robot system is a complex task – especially if cameras are used for visual guidance – since robotic manipulators are highly non-linear, electromechanical systems. Machine learning techniques can considerably help to realise a solution. Methods for model-free function approximation, such as Multilayer Perceptrons[1], Radial Basis Function networks[2] and Self-organizing Maps[3], were successfully applied to control tasks. However, these general learning methods may cause problems during training (e.g. not converging to the optimal solution) and they make global stability proofs for the controller difficult.

As alternatives to such singular networks built to solve the whole task at once approaches with ensembles [4], [5], Mixture of Experts [6] and piecewise linear approximators [7], [8], [9] were suggested. Latter are especially suitable for control tasks since sophisticated knowledge from control theory (important for stability proofs) and statistics (important for parameter estimation) about linear systems is applicable. The common idea behind piecewise linear techniques is to approximate a possible non-linear function by combining different linear models which are only valid for different but maybe overlapping regions of the input space. The mayor strategy is to achieve good global by good local approximation. The methods differ in aspects as: How is the region of validity of a linear model – also called model's domain – defined? How is a region's position and size in input space determined? How are the outputs of the linear models combined to the output of the whole network? Besides summarizing state of the art techniques, the following section will give and motivate new answers to these questions.

II. The new Approach

A. State of the Art Networks

Piecewise linear approximators can generally be formalized as:

$$\hat{\mathbf{y}}(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}) \Big(\mathbf{C}_i \mathbf{x} + \mathbf{b}_i \Big), \tag{1}$$

where $\hat{\mathbf{y}}$ denotes the estimated multi-dimensional output of the network for the given multi-dimensional input $\mathbf{x}$. The N coefficient matrices $\mathbf{C}_i$ and the bias vectors $\mathbf{b}_i$ are the parameters of the linear models. The scalar valued weighting factors $g_i(\mathbf{x})$ define the domains of the models $i \in \{1, \ldots, N\}$. Therefore, they 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 n training samples $T = \{(\mathbf{x}_k, \mathbf{y}_k)\}$ and a set of defined domains the parameters of the linear models can be trained. The goal is to minimize the approximation errors for the training sample which are located in the domain of one model. On the other hand, training samples which belong to another domain could have arbitrary bad approximation results. The parameters are typically determined by means of linear regression methods.

A common way (e.g. [7], [8] to define a model's domain is to place one center vector $\mathbf{c}_i$ in the input space

and to calculate the weighting factors according to some basis function, e.g. Gaussian:

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

Furthermore the weighting factors are typically normalized, so that a partition of unity of the input space is realised, i.e. that: $\sum_i g_i(\mathbf{x}) = 1$ and $\forall i : 0 \leq g_i(\mathbf{x}) \leq 1$. So, the idea is that each linear model has as its domain a radial or elliptic receptive field centered in $\mathbf{c}_i$.

It is part of the proposed strategies that such domains will overlap. In regions of the input space where this happens, i.e. where different weighting factors g_i are more or less equal, the output of the network will be an equally weighted, additive mixture of single linear models. That this strategy really improves the approximation of the target function depends essentially on the right choice of overlap between the domains. A good choice would result in a smooth transition between the different slopes of the linear models. The problem is that with weighting factors of the form of Eq. (2) (e.g. [7]), the overlap can only be controlled with the center $\mathbf{c}_i$ and the spread matrix $\mathbf{D}_i$ parameters. But due to the normalization, the domains will be distorted anyway. As discussed in [10], 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.

B. New Network with Sharply Bounded Domains

To overcome such imponderabilities, we propose a straightforward and effective way to define the domain of a linear model as:

$$g_i(\mathbf{x}) = \begin{cases} 1 & : \quad i = \arg_j \min d(\mathbf{x}, \mathbf{c}_j) \\ 0 & : \quad else \end{cases} . \qquad (3)$$

So, the output $\hat{\mathbf{y}}$ of the network will be the output of exactly one linear model. Which model is selected to compute $\hat{\mathbf{y}}$ depends solely on the distance[1] measure d of the input $\mathbf{x}$ to the center vectors $\mathbf{c}_i$: The model with the nearest center vector to $\mathbf{x}$ will estimate the real value of the approximated function.

This scheme defines a sharp boundary between domains. The outputs of linear models are not mixed. The boundary of two models lays halfway between their two center vectors. The input space is therefore divided into disjunct regions of which each is associated to exactly one linear model. This realises a partition of unity which is easier to control than with normalization used in the state of the art approaches.

[1]In the implemented models, we used the Euclidean distance.

Besides the parameters $\mathbf{C}_i$ and $\mathbf{b}_i$ of the linear model only the center vectors $\mathbf{c}_i$ must be learned from a given training set. In contrast to that, good spread matrices $\mathbf{D}_i$ must also be estimated from that data in [7] and [8].

C. State of the art Learning Algorithms

Given these networks, the essential problem of piecewise 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 in a set of regions where the target function can effectively be approximated with a linear model. So, the values for the parameters of the weighting factors g_i must be optimized. For the approaches of [7], [8], [9] these are basically[2] the center vectors $\mathbf{c}_i$ and spread matrices $\mathbf{D}_i$. As mentioned above, the parameters of the linear models are determined by means of linear regression methods. In [7] and [8] a weighted least square scheme was applied (see [11] for a discussion of this method). This is necessary since the domains of these methods define no sharp boundary and hence, a definite assignment of a training sample to one linear model is not possible. So, when the regression equation is formed the training samples are weighted by the factors g_i. By this means only training samples which belong most prominently to the domain of a model are effecting its parameters.

Before the parameters of the weighting factors g_i can be trained the question must be answered: How many linear models should the network contain? In this aspect, all approaches follow the strategy to incrementally increase the number of models if some criterion is fulfilled. For example in [7], new models are included to the network if for a given input $\mathbf{x}_k$ no weighting factor $g_i(\mathbf{x}_k)$ is larger than an user defined threshold[3]. In this approach, linear models are also removed from the network according to some similar 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.

While the linear model parameters can be trained in a straightforward supervised manner there is no such clear error criterion for the parameters of the weighting factors. The methods differ considerably in their approaches. A two-phase learning scheme is proposed in [8]: the center vectors are changed in Hebbian adaptation steps and by a gradient decent approach to minimize the least squares error function $J = \sum_{k=1}^{n}(\mathbf{y}_k - \hat{\mathbf{y}}(\mathbf{x}_k))^2$

[2]These methods have some additional parameters whose description would exceed the scope of this article.

[3]This comparison is made before the normalization of all g_i.

over the training set. In contrast to that, the authors of [7] prefer to minimize a so-called locally weighted error function which emphasizes that training samples only effects the domain's parameter they belong to. As discussed in [6], [12] the latter suits the overall strategy of mixture techniques better 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.

D. Learning Algorithm for the new Network

As the new model with its sharp boundaries between the model domains already introduces competition the learning algorithm emphasized this as well. It is a recursive scheme in which the training set is split into subsets belonging to local regions. Such subsets are assigned to new linear models. This splitting process is repeated as long as the local approximation error is not sufficient small and the regions contain enough training samples. Thereby each model added to the network will be trained, evaluated and – if necessary – optimized. A model will be optimized by being replaced by a number of new models which will occupy together the domain of the former one.

In detail: The learning algorithm starts with one linear model having the whole input space as its domain. The model's parameter $\mathbf{C}_i$ and $\mathbf{b}_i$ are trained with a least squares method using all training samples within its domain. The squared approximation errors for each training sample are computed. If their sum is larger than a user specified threshold and the number of samples used for training is sufficient for a splitting, new models will be added to the network and replace the currently optimized one. How the training set is split into subsets is described below in more details. To each subset a new model is assigned. For each of these new models the same procedure of training, evaluating and possible optimization will be performed.

A distinctive feature of the algorithm is how the training set is split and where the center vectors are placed. The training set of a model being optimized is split with the k-means algorithm [13], [14, in Sec. 5.9.1] in which the single training samples are weighted by the approximation error achieved by the model. The output of the weighted k-means algorithm is used as the center vectors for the new models replacing the former one. This procedure ensures that new models are inserted where approximation performance was bad. The normal k-means algorithm would place the center vectors driven only by the distribution of the samples in input space. The weighted version instead positions the vectors in clusters of training samples with high approximation error.

Although it remains the choice of the user how many models are generated in one splitting step we tested only splits in two halves. Another parameter of the learning algorithm is a positive scalar factor which is multiplied with the approximation errors before the weighted k-means algorithm is used. Tuning the factor means laying emphasis on centering the new models on samples with high errors or not. For noisy data the right choice of the factor is crucial since outliers could gain to much effect.

The mechanism of splitting and centering models at positions with high approximation error has the desired effect that more linear models are trained where the target function has high curvature. This is illustrated for some one dimensional synthetical test data in Fig. 1. Since the weighting factors define sharp boundaries between linear models also discontinuities can be handled. As visible in the figure, it is not guaranteed that the position of the discontinuity matches the boundary of two linear models. But for different test data it was always close by.

Fig. 1 also demonstrates that approximated functions will have discontinuities at the domain boundaries. This could be a problem for applications where the input is continuously changed and resulting jumps of the output can be dangerous. Other approaches are trying to solve this problem with soft domain definitions of the form of Eq. (2). Our algorithm will minimize this effect by inserting more models. Note that our network is computationally not less efficient just because it has more linear models. In our case only one linear model is evaluated while in the other approaches the outputs of all models must be computed.

The parameter which specifies the minimal number of training samples needed to perform a splitting is important to inhibit overfitting. If the value is too small the network will have too many linear models (and vice versa). Both will result in a poor ability of the network to generalize from the training set.

Overall, the algorithm has one discrete and two continuous parameters (the method in [8] has mainly due to its gradient decent scheme 13 parameters). These must be optimized on the training set. An implementation showed fast convergence (see III-A for numbers).

III. Experiments

To show the effectiveness of the our network and its learning algorithm two experiment are presented in this

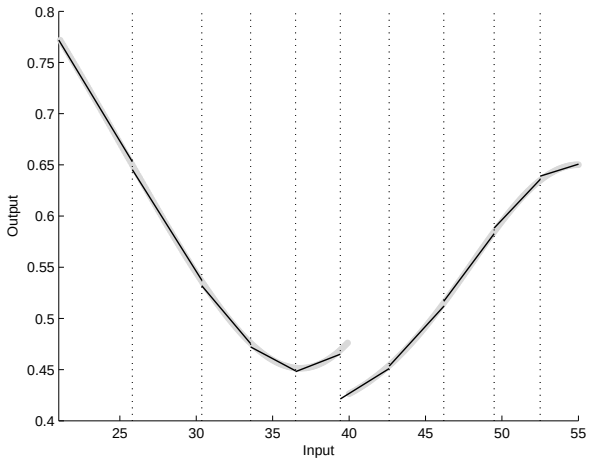

Fig. 1. Nonlinear function with discontinuity and its linear approximation. Dotted lines indicate boundaries between domains of models. Note, more linear models are used where curvature of function is high.

section. It describes one benchmark test and a real world robot application performed in our laboratory.

A. Mackey-Glass Benchmark

To compare our approach with one of the state of the art algorithm an experiment described in [8] with the chaotic Mackey-Glass differential equation was repeated. The experiment stems originally from [15] and defines the task to predict a value of the time series

$$x(t-1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \qquad (4)$$

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 to be approximated is defined as

$$x(t+6) = f\Big(x(t), x(t-6), x(t-12), x(t-18)\Big). \quad (5)$$

The network was trained on 1000 samples of f with t ranging from $t = 124$ to $t = 1123$. It was tested on an equal number of samples (between $t = 1124$ and $t = 2213$). Tab. I shows the achieved results of four different approximation methods: the Growing Multi Expert [8], the Orthogonal Least Squares [16], the Radial Basis Functions based on the Adaptive Fuzzy System [17] and our approach (beside the latter, all results are quoted from [8]). We implemented our algorithm in MatLab (R14) running on a Linux PC with a 3000 MHz CPU (1 GB RAM). The generation of a network for this benchmark test took always less than 30 sec..

Our methods showed superiority w.r.t. the achieved RMSE. Observable is that our solution needs more linear models than the other local linear method of [8]. One can assume that this is due to the different domain definitions. To approximate continuous functions our approach will generally need more linear models than methods with overlapping domains.

TABLE I
RESULTS OF BENCHMARK ON THE CHAOTIC MACKEY-GLASS DIFFERENTIAL EQUATION.

	# of hidden nodes	RMSE on test set
Our network	64	0.0029
GMN [8]	7	0.0091
RBF-AFS [17]	21	0.0128
OLS [16]	132	0.0163

B. Robot application

The new network was used as a controller with the task to position an arm robot w.r.t. image data. The visual servoing setup was the following: A Sony DFW-X710 digital color camera with a lens of 4.2 mm focal length was attached at the endeffector of a Stäubli RX90B arm robot. The goal was to drive the robot in a plane w.r.t. an object visible in the camera image laying ca. 45 cm beneath. The current and the target position of the object is known in image coordinates and must be transformed into a robot command to reach a new position. The servoing process is completed if the object is visible at the target position in the image. In our approach we specify robot commands in the world coordinate system of the robot. So basically, a transformation between two image coordinates and a robot command given in robot coordinates must be computed. Formally, the function f has to be found with:

$$\Delta\mathbf{R} = \begin{pmatrix} \Delta x^{(R)} \\ \Delta y^{(R)} \\ \Delta z^{(R)} \end{pmatrix} = f(x_c^{(I)}, y_c^{(I)}, x_t^{(I)}, y_t^{(I)}), \qquad (6)$$

where $x_{c,t}^{(I)}, y_{c,t}^{(I)}$ are the current and target image coordinates and the three dimensional $\Delta\mathbf{R}$ is the desired robot command ($\Delta x^{(R)}, \Delta y^{(R)}, \Delta z^{(R)}$ are the robot coordinates). In the implemented control scheme $\Delta\mathbf{R}$ is added to the current position of the robot in order to approach the target position.

By using the new network to learn the function f we omit the problem to model explicitely the relation between the coordinate system of the camera and the robot. It frees us from assumptions about the right camera model (e.g. a pinhole camera) or effects of lens distortion. It also reduces the effort to calibrate the built model to the actual setup.

In order to learn the function f a set of training samples were compiled by driving the robot stepwise

to positions on a plane where the object was visible in the camera image. At each step the position of the robot and the position of the object in image coordinates were measured. It was ensured that the object was presented in all parts of the image. With this data a set of 330 training samples were generated. The informations of two adjacent positions from the sample run were taken to define $x_{c,t}^{(I)}, y_{c,t}^{(I)}$ and $\Delta\mathbf{R}$: The image data of one was taken as the current, the other as target position and the difference between their robot positions was taken as the desired output $\Delta\mathbf{R}$. Given this data, a network was trained and optimized w.r.t. the parameters of the learning algorithm according to the following equation:

$$\hat{f}(x_c^{(I)}, y_c^{(I)}, x_t^{(I)}, y_t^{(I)}) = \sum_{i=1}^{N} g_i(x_c^{(I)}, y_c^{(I)}) \Big(\mathbf{C}_i \Delta\mathbf{I} + \mathbf{b}_i \Big), \tag{7}$$

with $\Delta\mathbf{I} = \left(x_c^{(I)} - x_t^{(I)}, y_c^{(I)} - y_t^{(I)} \right)^T$. This special usage of Eq. (1) means that the weighting factors are using only absolute image information while the linear models are relating a distance in image space to a distance in robot space. It is motivated by the fact that one can assume a linear relation between moving in a plane and observing the resulting displacement of an object in the image plane. Due to the contortion between the two planes and effects of lens distortions this assumption will only be valid in local regions and depend on the absolute position of the object in the image.

The performance of the generated network was evaluated by using it to drive the robot to a similar set of positions as in the training run. The control scheme achieved qualitatively good results. Covering distances of about 50 pixel in one step the controller achieved normally a misplacement of less than 2 pixels. With the object recognition methods we used this is absolutely sufficient since measured positions of an object in a static test scenario vary to the same degree. Furthermore a quantitative evaluation of the test run showed that the root mean squared error between target position and actually achieved position in robot space was $RMSE = 1.06\,mm$ (with standard deviation $\sigma = 1.1\,mm$).

An advantage of the approach to learn and not to model explicitely the transformation function f is that it can be extended more easily to more complex servoing tasks. We already started to drive the robot – not just in a plane – but freely in 3D by using two cameras. Therefore, only the input vector of f was extend by the current and the target position of the second camera and training positions in 3D were sampled differently.

IV. Discussion

We proposed a new network and learning algorithm for piecewise linear function approximation in the context of system control. The development was driven by two goals: Good global approximation should be achieved by competition between local linear models and the insertion of new local models into regions of the input space where the error is high. On the other hand, simplicity of the network and algorithm should be achieved. This is important for interpretability of the network (necessary for stability proofs needed for controllers) and the costs to optimize the parameters of the training algorithm.

Competition between models is reflected in the fact that these are only trained with samples from their sharply defined domains. Since a training set is – before it will be clustered – weighted with the approximation errors more linear models are placed where the target function has high curvature. Good interpretability of the network is achieved by defining the domains without any overlap. As the algorithm has a comparably small number of parameters the costs for training an effective network are quite manageable.

In our future work we will tackle different topics: We plan to optimize the definition of the domain and aspect of the learning algorithm. Since the boundary between two domains is the middle between their two center vectors shifting one vector affects the other's domain. This is against the idea of distinctive local learning. We are starting to test domain definitions which places vectors at the boundary (similar to support vectors in SVMs). The algorithm should also be extended to remove or combine linear models when they do not improve global approximation. This is desirable since a network with too many models will tend to data overfitting.

Furthermore the plan is to transfer our batch learning scheme to one with an online training of the network. And, last but not least, we want to apply knowledge from control theory to obtain a stability proof for the network used as an adaptive controller.

V. 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] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks: Controllability and stabilization," *IEEE Transactions on Neural Networks*, vol. 4, pp. 192–206, Mar. 1993.

[2] C. Baroglio, A. Giordana, M. Kaiser, M. Nuttin, and R. Piola, "Learning controllers for industrial robots," *Machine Learning Journal*, vol. 23, pp. 221–249, / 1996.

[3] J. A. Walter and K. J. Schulten, "Implementation of self-organizing neural networks for visuo-motor control of an industrial robot," *IEEE Transactions in Neural Networks*, vol. 4, pp. 86–95, Jan. 1993.

[4] A. J. Sharkey, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[5] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

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

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

[8] 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.

[9] 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 the Sixth International Conference on Neural Information Processing (ICONIP'99)*, (Perth, Australia), pp. 520–525, Nov. 1999.

[10] 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.

[11] W. Cleveland and C. Loader, "Smoothing by local regression: Principles and methods."

[12] 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.

[13] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. IT-28, pp. 129–137, Mar. 1982.

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

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

[16] 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.

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