ENGINEERING MECHANICS 2004
NATIONAL CONFERENCE
with international participation
Svratka, Czech Republic, 10 - 13 May 2004

APPROXIMATION OF WALKING ROBOT STABILITY MODEL

Jiří Krejsa[*], Robert Grepl*, Stanislav Věchet[+]

Abstract: *The paper compares global and local approximation methods used for walking robot stability model. Global approximators are represented by feedforward multilayer neural network (FFNN) trained by gradient method; local approximators are represented by Locally Weighted Regression (LWR) and Receptive Field Weighted Regression (RFWR) methods. Global approximators try to learn global non-linear function which fits all the training data (minimizes training error), while local approximators use spatially limited data in query point neighborhood to generate appropriate response. Various aspects of used approximation methods are discussed (precision, robustness, computational and memory requirements).*

1. Introduction

Our research activities are presently focused on modelling of quadruped walking robot. The properties of designed model has to be different according to different requirement posed on model. Paper [3] describes the design and characteristics of dynamic model of stability built in Matlab/Simulink/SimMechanics.

There are two main reasons why it is not possible to use this model for real-time control of the robot:

- long time of simulation (in seconds) on ordinary PC
- the energy consumption - the control algorithm has to work on microcontroler (not on PC)

The main goal is to build an approximation model which can be used instead of SimMechanics model. The complexity of algorithm should allow implementation on microcontroler with real-time speed of computation.

2. Materials and methods

2.1. Scheme of approximation model

We consider the simple type of gait, often called static gait. Three legs are in contact with the ground permanently, fourth is changing its position. The scheme of model is shown on Fig. 1.

[*] Ing. Jiří Krejsa, PhD., Ing. Robert Grepl: Institute of Thermomechanics, CAS, Mechatronics Centre Brno, Technická 2, 616 69, Brno, tel: +420 54114 2885, email: jkrejsa@umt.fme.vutbr.cz
[+] Ing. Stanislav Věchet, Institute of Mechanics, Mechatronics and Biomechanics, FME, BUT, Technická 2, 616 69, Brno

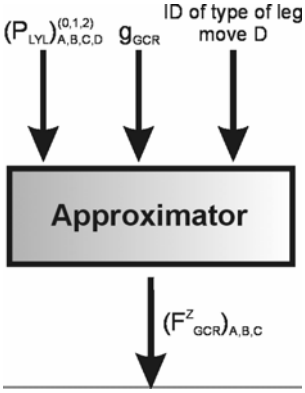

Fig. 1. Model scheme

The positions of stable three legs and the vector of gravity (can have various direction with respect to local robot coordinate system) form the inputs into the model. Last input called "ID" describes the type of moving trajectory of leg selected from certain small set.

During the first numerical experiments we used simplified schema of model. Motion in plain was considered only, fourth leg makes no movement. The reason was the size of state space of model shown on Fig. 1. In spite of such simplification, the model is better than the very simple one actually used in learning algorithms (triangle of stability).

The extension of simplified model close to general one (Fig. 1) can be performed. When using neural networks the segmentation of the state space would be an answer, however, one would expect the local approximators to deal with the general model itself.

2.2 Neural networks

Traditional feed forward layered neural networks with single hidden layer were used. Training algorithms included well known Levenberg-Marquardt algorithm (LM) and Broyden, Fletcher, Goldfarb, and Shanno update (BFGS) as representatives of quasi-Newton algorithms and Scale Conjugate gradient (SCG) algorithm as representative of conjugate gradient methods. We consider the methods generally well known, detailed information can be found in e.g. [2].

2.3 Locally weighted regression (LWR)

LWR is simple approximation method with locally linear model, based on the least squares method, introduced in [1], which minimizes simple criterion

$$C = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 \tag{1}$$

where n is input dimension, y_i represents given output, $\hat{y}_i$ is the fitted output. A linear model is defined as an equation with linear coefficients $y = p_1 x + p_2$, where p_1, p_2 are unknown coefficients. For n given points we solve a system of n linear equations with two unknowns. In matrix form, linear models are given by the equation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} \tag{2}$$

where $\mathbf{y}$ is the output matrix, $\mathbf{X}$ is the input matrix and $\boldsymbol{\beta}$ is matrix of unknown parameters. The equation can be solved as

$$\mathbf{X}^T\mathbf{y} = \left(\mathbf{X}^T\mathbf{X}\right)\boldsymbol{\beta} \tag{3}$$

$$\boldsymbol{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

In Locally weighted regression method, the prediction depends on distance between query point and given points in close neighborhood. Typical distance function is Euclidean distance

$$d_E\left(\mathbf{x},\mathbf{q}\right) = \sqrt{\sum_i \left(\mathbf{x}_i - \mathbf{q}\right)^2} \tag{4}$$

where **q** is the query point, $\mathbf{x}_i$ is the *i*th component of vector **x**. A weighting function or kernel function *K()* is used to calculate a weight for each data point. Typical weighting function is a Gausian

$$K\left(d_E\right) = e^{-\left(Dd_E\right)} \tag{5}$$

where d_E is the euclidean distance and *D* is a distance metric. Parameter *D* determines the width of the gausian kernel.

The weight for each stored data point is

$$w_i = \sqrt{K\left(d\left(\mathbf{x}_i, \mathbf{q}\right)\right)} \tag{6}$$

The weights form a diagonal matrix **W** with diagonal elements $\mathbf{W}_{ii}=w_i$. The weighted input **Z** is computed as $\mathbf{Z} = \mathbf{W}\,\mathbf{X}$. Equation no.3 is solved for $\boldsymbol{\beta}$ using the new variable

$$\mathbf{Z}^T\mathbf{y} = \left(\mathbf{Z}^T\mathbf{X}\right)\boldsymbol{\beta} \tag{7}$$

$$\boldsymbol{\beta} = \left(\mathbf{Z}^T\mathbf{X}\right)^{-1}\mathbf{Z}^T\mathbf{y}$$

The only open parameter in equation 5 is the distance metric *D*, which can be optimized by leave-one-out cross validation: for all data for *i* to *n*, temporarily exclude $\{x_i, y_i\}$ from training data, compute LWR prediction $\hat{y}_i$ and compute error $e = \sum_{i=1}^{n}\left(y_i - \hat{y}_q\right)^2$. This algorithm is used with different distance metrics *D* and *D* with minimal error is chosen as optimal.

2.4. Receptive Field Weighted Regression (RFWR)

Receptive Field Weighted Regression algorithm (RFWR) automatically creates structures necessary for solving regression tasks. The algorithm approximates data using spatially limited linear models. The size and shape of receptive fields of particular linear models, as well as parameters of local linear models are adapted independently. The algorithm was first introduced in 1997 [4].

The goal of RFWR is to create a system of receptive fields for incremental function approximation. The prediction of output $\hat{\mathbf{y}}$ for input **x** is calculated as normalized weighted sum of particular receptive fields predictions $\hat{\mathbf{y}}_k$:

$$\hat{\mathbf{y}} = \frac{\sum_{k=1}^{K} w_k \hat{\mathbf{y}}_k}{\sum_{k=1}^{K} w_k} \tag{8}$$

Weight w_k corresponds to the activation of corresponding receptive field and is determined from the size and shape of receptive field, which is characterized by kernel function. In order to obtain smooth approximation the suitable kernel function must be smooth and symmetric, eg. Gaussian kernel:

$$w_k = \exp\left(-\frac{1}{2}\left(\mathbf{x}-\mathbf{c}_k\right)^T \mathbf{D}_k \left(\mathbf{x}-\mathbf{c}_k\right)\right), \text{ where } \mathbf{D}_k = \mathbf{M}_k^T\mathbf{M}_k$$

which gives parameters of receptive field by its locating in space $\mathbf{c}_k$ and defining the size and shape of receptive field through positive definite distance matrix $\mathbf{D}_k$. This matrix is further represented by upper diagonal matrix $\mathbf{M}_k$.

The relation between input and output data in each receptive field is given by simple parametric function. One possible way is the use of locally linear models:

$$\hat{\mathbf{y}}_k = \left(\mathbf{x} - \mathbf{c}_k\right)^T \mathbf{b}_k + b_{0,k} = \tilde{\mathbf{x}}^T \boldsymbol{\beta}_k, \ \tilde{\mathbf{x}} = \left(\left(\mathbf{x} - \mathbf{c}_k\right)^T, 1\right)^T. \tag{9}$$

The learning algorithm adapts the parameters of receptive fields ($\mathbf{M}_k$, $\mathbf{c}_k$, $\boldsymbol{\beta}_k$) for each field independently. Receptive fields are added and pruned according the need, therefore the number of receptive fields corresponds with the problem to be solved.

Learning algorithm therefore consists of three stages:

1. adaptation of linear model parameters $\boldsymbol{\beta}_k$
2. adaptation of distance matrix $\mathbf{D}_k$
3. adding and pruning of receptive fields

Ad 1)

Learning of linear model parameters $\boldsymbol{\beta}$ is simple because the problem is linear. Using *n* pairs of training data (input $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, corresponding output $\mathbf{Y}$) and weight matrix $\mathbf{W}$ with corresponding weights on main diagonal the parameter $\boldsymbol{\beta}$ can be obtained by weighted regression:

$$\boldsymbol{\beta} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{Y} = \mathbf{P} \mathbf{X}^T \mathbf{W} \mathbf{Y} \tag{10}$$

To avoid explicit inversion of the matrix one can use recursive least square method. Adaptation rule for $\boldsymbol{\beta}$ for given data element $(\mathbf{x}, \mathbf{y})$ is then:

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n + w \mathbf{P}^{n+1} \tilde{\mathbf{x}} \mathbf{e}_{cv}^T, \text{ where}$$

$$\mathbf{P}^{n+1} = \frac{1}{\lambda}\left(\mathbf{P}^n - \frac{\mathbf{P}^n \tilde{\mathbf{x}} \tilde{\mathbf{x}}^T \mathbf{P}^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T \mathbf{P}^n \tilde{\mathbf{x}}}\right), \ \mathbf{e}_{cv} = \mathbf{y} - \boldsymbol{\beta}^{nT} \tilde{\mathbf{x}}, \ \tilde{\mathbf{x}} = \left(\left(\mathbf{x} - \mathbf{c}_k\right)^T, 1\right)^T \tag{11}$$

This formula already contains the forgetting factor λ necessary for sequential forgetting of the past data elements contributions, made when the distance matrix $\mathbf{M}$ is not yet fully learned (change in matrix $\mathbf{M}$ made during learning changes also the weight w)

Ad 2)

The change of size and shape of receptive field is made via adaptation of distance matrix $\mathbf{M}$, in particular through steepest descent algorithm used for error function J.

$$\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha \frac{dJ}{d\mathbf{M}}, \text{ where } \alpha \text{ is learning parameter.} \tag{12}$$

Error function J is obtained from weighted mean of square error:

$$J = \frac{1}{W}\sum_{i=1}^{n} w_i \left\| y_i - \tilde{y}_i \right\|^2 , \tag{13}$$

where $\hat{y}_i$ is predicted output, n is number of receptive fields and $W = \sum_{i=1}^{n} w_i$

Error function defined this way leads to the overlearning problems. Therefore one can use leave-one-out cross validation and further introduce penalty term. Final form of the error function is then:

$$J = \frac{1}{W}\sum_{i=1}^{n} \frac{w_i \left\| y_i - \tilde{y}_i \right\|^2}{\left(1 - w_i \tilde{\mathbf{x}}_i^T \mathbf{P} \tilde{\mathbf{x}}_i\right)^2} + \gamma \sum_{i,j=1}^{n} D_{i,j}^2 \tag{14}$$

Ad 3)

Newly created receptive field is added whenever given training pair $(\mathbf{x}, \mathbf{y})$ does not activate existing receptive fields for more than w_{gen}. The center of newly created receptive field is set equal to the input of the training fact $\mathbf{c} = \mathbf{x}$. Distance matrix $\mathbf{M}$ is set into predefined value and other parameters are set to zero.

The algorithm also contains pruning of excessive receptive fields. If certain receptive field overlaps another field (training fact activates two receptive fields more than w_{prune}) then receptive field with higher determinant of distance matrix $\mathbf{D}$ is pruned.

3. Numerical simulations

3.1 Simulation details

Neural networks were tested using Matlab implementation (Neural Network Toolbox). Above mentioned algorithms were used, with Levenberg Marquardt given the best compromise between computational and memory requirements versus training convergence speed and precision. Layered networks contained 4 - 30 neurons in hidden layer, networks with 8 and 22 hidden neurons were selected for further testing. Other training parameters were set as follows:

transfer function – hidden layer	hyperbolic tangent sigmoid
transfer function – output layer	linear
training cycles	1000
weights initialization	random (-1, 1)

Tab 1. NN parameters

LWR algorithm was implemented in Matlab. The only parameter of the method to set is the kernel width D (eq. 5), which was set to 100.

RFWR algorithm was implemented in Delphi 7 and initially tested on simple single variable function approximation task. Results were encouraging and algorithm was further used for the task at hand. The parameters of the RFWR algorithm were set as shown in Tab. 2:

w_{gen}	=	0.1
w_{prune}	=	0.99
λ_{init}	=	0.999
λ_{final}	=	0.9999
λ_{delta}	=	0.99999
α_{init}	=	200
γ	=	1e-10

Tab.2. RFWR algorithm parameters

3.2 Algorithms comparison

Test results of the simulations for various approximators are shown in Figures 2. - 5. Data used for training were excluded from the training set, however, they also cover the complete input space differing in the grid resolution.

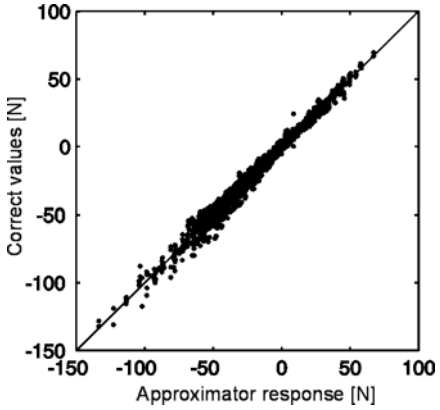

Fig. 2. Results for NN 6-8-1

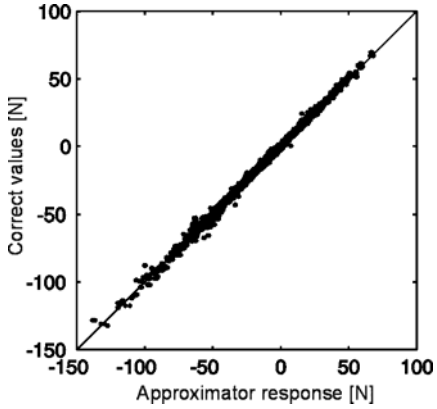

Fig. 3. Results for NN 6-22-1

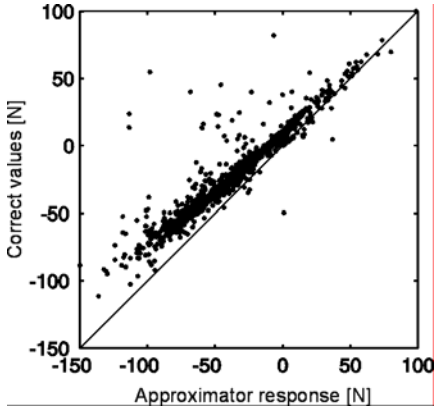

Fig. 4. Results for LWR

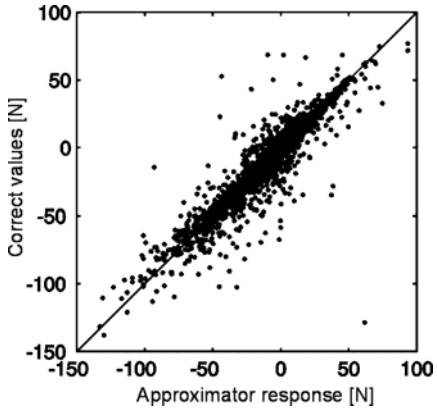

Fig. 5. Results for RFWR

The comparison of methods errors and computational requirements is shown in Tab. 3. *MSE* stands for Mean Square Error, *Max E* stands for Maximum error, *Don't know responses* is the number of responses when approximator couldn't give an answer, *Training time* gives the total time required for training, *Total parameters* is the total value of adjustable

parameters of given approximator (this value influences total memory requirements), *Response parameters* is the number of parameters used when response for single input fact is calculated.

Approximator	**NN 6-8-1**	**NN 6-22-1**	**LWR**	**RFWR**
MSE	580×10^{-6}	130×10^{-6}	819×10^{-6}	1.02×10^{-3}
Max E [N]	18.49	12.56	48.76	36.01
Don't know responses	-	-	95 (0.35 % cases)	47 (0.20 % cases)
Training time [h:min]	2:00	7:10	4:03	3:11
Total parameters	65	177	189000	56784
Response parameters	65	177	variable	variable

Tab. 3. Algorithm comparison

As we can see, the error on test data is higher for LWR and significantly higher for RFWR than test error produced by neural network approximator. Learning speed is about the same, however the memory requirements are higher, even for larger neural networks. LWR algorithm must store all the data points in memory (27000 input vectors). RFWR algorithm generated 728 receptive fields which gives number of parameters 56784 (each receptive field has at least center matrix – 6 parameters and D and P matrices of 6x6). Neural networks must store only synaptic weights (total number of parameters depend on number of hidden neurons), e.g. network 6-8-1 has 65 parameters only. During the query response the situation is different for global and local approximators. Neural network must use all its parameters, while local approximators use only the neighborhood points in LWR case and activated receptive fields in RFWR case.

4. Conclusions

LWR algorithm main advantage is its extremely simple implementation. Algorithm is also very fast, the only training it requires is finding the optimal kernel width. The main drawback is the memory requirement – the algorithm requires keeping all the data points in memory. Unfortunately this makes the algorithm unusable for large data sets.

First advantage of RFWR algorithm over neural network approach is clear: the RFWR approximator can give "don't know" result. When number of receptive fields activated for given test input is zero the system returns corresponding flag. This is contrary to neural networks which gives some output for arbitrary input. LWR algorithm can produce the "don't know" answer as well, when close neighborhood doesn't contain sufficient information (number of close data points).

Furthermore, RFWR algorithm is in principle very robust against partial overtraining (as any other local approximator). That means that if new training data are presented to the RFWR after end of the training, it simply learns new data and already trained data remains intact. This is again contrary to neural network when further training on new data can destabilize learned global function. LWR algorithm can not be overtrained in principle.

However, looking at the data on this concrete application one can see that neural networks beat local approximators in almost all aspects. High error values on test data could be probably improved with better settings of RFWR learning parameters, this work is currently in progress. LWR results can hardly be improved – distant function change nor kernel function change will improve the overall results significantly. Advantages of the local approximators mentioned above are according to us of high value and the algorithms (mainly RFWR and its successors) is worth of further exploration, but currently the neural network is the best answer for given task.

Acknowledgement

This work was supported by Czech Ministry of Education through project MSM 262100024 "Research and development of mechatronic systems".

References

1. Atkeson C.G., Moore A.W., Schaal S.: Locally Weighted Learning, Technical Report, ATR Human Information Processing Laboratories, Japan, 1996
2. Demuth H., Beale M. Neural Network Toolbox User's Guide, Mathworks, Inc., 2002
3. Grepl, R.: Dynamic model for stability control of walking robot, Colloquium Dynamics of Machines 2004, Prague, February 10-11, Czech Republic, 2004
4. Schaal, S. & Atkeson, C. G.. "Receptive Field Weighted Regression." Technical Report TR-H-209, ATR Human Information Processing Laboratories, Japan, 1997