MODEL INVERSION USING FUZZY NEURAL NETWORK
WITH BOOSTING OF THE SOLUTION

Paolo Mercorelli1 §, Mirko Nentwig2
1Leuphana University of Lueneburg
Institute of Product and Process Innovation
Volgershall 1, D-21339 Lueneburg, GERMANY
2Audi AG
Department of Hardware-in-the-Loop Functional Testing
D-85045 Ingolstadt, GERMANY

Abstract: Neural networks are a very effective and popular tool for modeling. The inversion of a neural network makes possible the use of these networks in control problem schemes. This paper presents an inversion strategy based upon a feed-forward trained local linear model tree. The local linear model tree is realized through a fuzzy neural network.

AMS Subject Classification: 92B20, 03B52, 90B15
Key Words: neural networks, fuzzy logic, networks model

1. Introduction and Motivation

In some of application problems, physics-based models may be used, but these are very complex and must be simplified for use, which degrades the model. Neural networks are another option for modeling complex systems. They are relatively straightforward systems but they may be appropriate even for highly complex modeled problems. The purpose of this work is to show that neural
networks can be applied successfully also for inversion of model in control problems. An application in the automotive control field of a model inversion using such kind of approach is presented in [1] and in [2]. Thanks to a boosting procedure, the algorithm can be applied also in the presence of noise. The paper is organised as follows: Section 2 introduces the concept of LOLIMOT and the general problem of a model inversion. Section 3 is devoted to the algorithm proposal in Section 4 some aspects of it are discussed. The conclusions close the paper.

2. Model Inversion

The inversion problem in neural networks has attracted many researchers and mathematicians. This is a difficult problem which involves the inversion of the nonlinear membership functions. This procedure applies the LOLIMOT algorithm [3], which is based upon Neural-Fuzzy models of Takigi Sugeno type. During the execution of this algorithm, a ”divide and conquer strategy” is applied to the modeling problem, so that the major problem is split into smaller ones. The basic network structure of a local linear neural fuzzy model is depicted in Fig. 1. Every neuron consists of a local linear model (LLM) and a validity function \( \Phi \) defining the validity of the LLM within the input space. The local linear model output is defined by:

\[
\hat{y}_i = w_{i0} + w_{i1}u_1 + w_{i2}u_2 + \ldots + w_{ip}u_p,
\]

(1)

where \( w_{ij} \) is the LLM parameter at each neuron \( i \).

If the validity functions are chosen as normalized Gaussians, then it follows:

\[
\sum_{i=1}^{M} \Phi_i(u) = 1 \quad \text{and} \quad \Phi_i = \frac{\mu_i(u)}{\sum_{j=1}^{M} \mu_j(u)},
\]

where the membership function \( \mu_i \) is

\[
\mu_i(u) = \exp \left( -\frac{1}{2} \left( \frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2} \right) \right) + \exp \left( -\frac{1}{2} \left( \frac{(u_2 - c_{i2})^2}{\sigma_{i2}^2} \right) \right) + \ldots + \exp \left( -\frac{1}{2} \left( \frac{(u_p - c_{ip})^2}{\sigma_{ip}^2} \right) \right).
\]

(2)

To achieve an inversion, we develop an algorithm which allows us to obtain the required model input \( u_r \), depending on the desired model output \( y \) and the other model inputs \( u \). Fink, Toepfer and Isermann [4] offer some strategies in their analysis of the inversion of nonlinear models:
- Inverse access by numerical inversion

Only one model of the nonlinear function is created and used for standard and inverse access. The inverse access equals a numerical inversion and requires the application of optimization methods to determine the input.
for the requested output.

- Data driven generation of an inverse model
  
  A model for inverse access is created in addition to the model for standard access.

- Analytical inversion of models
  
  A direct inversion of the forward trained model is applied. Hence, it is an advantage to use model architectures which allow the direct calculation of the inverse model using its own parameters. The developed algorithm applies an analytical/numerical inversion of a given local linear model structure, and is explained below. The following constraints are required to set up the algorithm:

  - An existing forward trained local linear model tree of the process is available.
  
  - The expected model output $y$ is known.
  
  - There are existing input values for those inputs upon which $y$ is dependent.

### 2.1. The Validity Functions Issue

Consider the model output function $\hat{y}$, with $M$ local linear model $\mathbf{u} = [u_1, ..., u_p]$ inputs

$$\hat{y} = \sum_{i=1}^{M} (w_iu_0 + w_iu_1 + w_iu_2 + ... + w_iu_p) \Phi_i(\mathbf{u}),$$

(3)

the validity function

$$\Phi_i = \frac{\mu_i(\mathbf{u})}{\sum_{j=1}^{M} \mu_j(\mathbf{u})},$$

(4)

and the membership function

$$\mu_i(\mathbf{u}) = \exp \left( -\frac{1}{2} \left( \frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2} \right) \right) +$$
$$\exp \left( -\frac{1}{2} \left( \frac{(u_2 - c_{i2})^2}{\sigma_{i2}^2} \right) \right) + ... + \exp \left( -\frac{1}{2} \left( \frac{(u_p - c_{ip})^2}{\sigma_{ip}^2} \right) \right).$$

(5)
A difficulty is that, due to the exponential quadratic nonlinearity, the model is not invertible. Hence, it is necessary to convert the functions into a linear type, as shown in Fig. 2. The membership function is split into a spline function that consists of the linear functions.

\begin{equation}
\mu_{ir} = \begin{cases} 
  \frac{k_\sigma}{1.6 \cdot \sigma_r} (u_r - c) + 1, & -\frac{1.6\sigma_r}{k_\sigma} + c \leq u_r \leq c, \\
  \frac{k_\sigma}{1.6 \cdot \sigma_r} (u_r + c) + 1, & c \leq u_r \leq \frac{1.6\sigma_r}{k_\sigma} + c.
\end{cases}
\end{equation}

Using the function defined in equation (6), it is now possible to invert the local linear model tree.

3. Algorithm for the Inversion of the LLM

We apply the following algorithm to invert the model:

- Calculate the LLMs represented in equation (3), omitting the required input variable $u_r$. That is, calculate the LLMs with the available input data until there only remains a linear equation depending on one input,
e.g., \( y_i = w_r \cdot u_r + u_{calc} \). To be more comprehensible, if \( \hat{y}_i = w_{i0} + w_{i1}u_{i1} + w_{i2}u_{i2} + w_{i3}u_{i3} \) and \( u_{i1} \) is required, then \( y_i = w_{i1}u_{i1} + u_{calc} \), with \( w_ru_r = w_{i1}u_{i1} \) and \( u_{calc} = w_{i0} + w_{i2}u_{i2} + w_{i3}u_{i3} \).

- Calculate the membership functions represented in equation (5), omitting the required input variable \( u_r \). The membership function of the LLMs is calculated with the available input data to the extent possible. Since the nonlinear term depending on \( u_r \) is omitted, the membership function is a constant number. To be more precise,

\[
\mu_c(u) = \exp\left(-\frac{1}{2}\left(\frac{(u - c_{i2})^2}{\sigma_{i2}^2}\right)\right) + \ldots + \exp\left(-\frac{1}{2}\left(\frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right)\right).
\]

\( i = 1, \ldots, q \)

Then, the input \( u_r \) is reconsidered in the final membership function and the following expression is obtained,

\[
\mu_i(u) = \exp\left(-\frac{1}{2}\left(\frac{(u_r - c_{ir})^2}{\sigma_{ir}^2}\right)\right) + \mu_c.
\]

- Create the linear membership function for the required input as from equation (6).

- Partition the input space \( u_r \).

The input space of \( u_r \) is partitioned into \( q \) search intervals, which are used in the later estimation of \( u_r \). Every interval describes the validity of half of the local linear model. Thus, the input space of every LLM is divided into two intervals. This is necessary due to the structure of the new linear membership functions, because they consist of two linear functions as mentioned above. For every interval, a "left function" and a "right function" are considered (Fig. 5). For the sake of brevity, equation (6) is represented as the following:

\[
\mu(u_r)_{i,r} = \left\{ \begin{array}{ll}
\mu(u_r)_{i,r,1} \\
\mu(u_r)_{i,r,2}
\end{array} \right.
\]

- In the following loop, consider every interval for a possible solution of \( u_r \).

- Determine which of \( i \) membership functions \( \mu(u_r)_{i,r} \) are valid for the currently considered interval, by checking every spline. \( \mu(u_r)_{i,r,1} \) is valid if

\[
\mu(interval_{left})_{i,r,1} \geq 0 \land \mu(interval_{right})_{i,r,1} \leq 1
\]
\( \mu(u_r)_{i,r,2} \) is valid if
\[
\mu(\text{interval}_{\text{left}})_{i,r,2} \leq 1 \land \mu(\text{interval}_{\text{right}})_{i,r,2} \geq 0,
\]
where \( \land \) indicates the "and" logical function.

- Use the valid membership spline functions to create validity functions for each local linear model. Taking the previously calculated part of the membership function \( i, \mu_{i,\text{calc}} \) as in (8), and sum it with the valid linear spline membership function \( \mu(u_r)_{i,r,\{1,2\}} \), where \( \mu(u_r)_{i,r,\{1,2\}} \) represents a valid spline membership function within the range of functions. This yields:
\[
\mu(u_r)_i = \mu(u_r)_{i,r,\{1,2\}} + \mu_{i,\text{calc}}
\]
and
\[
\Phi(u_r)_i = \frac{\mu(u_r)_i}{\sum_{j=1}^{M} \mu(u_r)_j}.
\]
If there are no valid linear membership functions for a local linear model, then the model will not be considered for further actions.

- Initially create the output function for every local linear model by multiplying its validity function with the local linear model function:
\[
\hat{y}(u_r)_{\text{LLM},i} = \hat{y}(u_r)_i \cdot \Phi(u_r)_i.
\]
Next, sum the output functions to create the model output:
\[
\hat{y}(u_r) = \sum_{i=1}^{M} \hat{y}(u_r)_{\text{LLM},i}.
\]
Finally, equate the model output function to the desired model output value, and solve the resulting equation with respect to the variable \( u_r \):
\[
y = \hat{y}(u_r).
\]

- Verify that the calculated \( u_r \) is inside the currently considered interval.
\[
\text{interval}_{\text{left}} \leq u_r \leq \text{interval}_{\text{right}}.
\]
If so, accept it as one possible solution. If not, disregard it.
Due to the structure of the validity functions, an inversion of the model is possible only within the input space of the required variable. Beyond these borders, the model input will drift to zero, which is comparable to the normal LOLIMOT behavior. That is, once a work nominal point is chosen, the inversion is possible within the input domain. The worst case is if the working nominal point is close to the border of the domain. If so, a more suitable division of the input space is needed, as is described in the next section.

3.1. Boosting

With a continuous system, it is possible to accelerate the algorithm’s runtime behavior by reducing the time needed to select the validity functions during the inversion. This makes it necessary to perform some off-line calculations on the linear validity functions, which otherwise remain unchanged during the execution of the inversion. For every linear validity function, two points are calculated and saved in a look-up table: the starting point \( p_s \) and the point \( p_e \) where the function intersects the input domain axis. This information is used at runtime to pre-select the relevant validity functions; therefore, a region of interest (ROI) around the previously calculated desired input value \( u_r(k - 1) \) has to be defined as an interval, e.g.,

\[
ROI(u_r(k - 1)) = [u_r(k - 1) - c \cdot u_r(k - 1) \quad u_r(k - 1) + c \cdot u_r(k - 1)],
\]

where \( c \) is a parameter describing the size of the region. If the function crosses, starts, or ends in the ROI, the validity function should be considered.

A logic validity function (LVF) can be defined as follows:

\[
LVF = (p_s \leq ROI_{max} \lor p_s \geq ROI_{min}) \lor (p_e \leq ROI_{max} \lor p_e \geq ROI_{min}) \lor \quad (p_s > ROI_{max} \land p_e < ROI_{min}) \lor (p_e > ROI_{max} \land p_s < ROI_{min}),
\]

where "\lor" and "\land" indicate the "or" and the "and" logical functions, respectively. If the variable LVF assumes a value equal to 1, then the validity function should be considered; if the variable LVF assumes a value equal to 0, then the validity function should be not considered. The proposed boosting algorithm is similar to the algorithms used to solve clipping problems in computer graphics, in which the ROI states the camera field of view.
4. Analysis of the Algorithm

4.1. Stability

The algorithm is mostly stable; however, due to the limited character of the linear validity functions, it is possible that no solution will be found in the border regions of the model. In that case, there are two possible solutions:

- The first solution is to use a more complex approximation of the validity function, e.g., a linear spline consisting of $n$ line segments. This will increase the computational effort. In fact, each line segment represents a single interval which must be considered to solve the inversion problem. Since the validity function is symmetric, the effort is increased by a factor of two. Applying the proposed boosting method will reduce the complexity.

- Another solution is to adjust the zeros of the linear approximation function so that the function will cover a larger region of the input domain. This will reduce the accuracy of the estimated input value, but it keeps the computational costs low.

4.2. Ambiguities

Since the algorithm solves a squared equation, it is possible to obtain two solutions for each of the $q$ intervals if the solutions are valid. In the worst case this results in $2 \times q$ solutions. Therefore, we introduce a decision criterion to select the correct input value. If the system is continuous, as it is in most real applications, this decision criterion could be based on the previous input value $u_r(k - 1)$. Other criteria could also be used.

4.3. Accuracy

The accuracy of the inversion is mainly influenced by the linear validity function. Therefore, a linear/linear spline validity function will lead to more accurate results compared with the simplest linear function. As mentioned above, this will increase the computational complexity, so the best compromise between speed and accuracy has to be found. Boosting can be used to reduce these negative effects. By using the simplest validity functions, with one linear spline per side, a difference of 10 - 15 percent between the predicted input value and
the real input value can occur in the worst case. However, the result can be improved by applying a numerical optimization process.

5. Conclusions and Outlook

This paper describes a powerful algorithm for the inversion of local linear model trees at runtime. The algorithm was analyzed in terms of stability and accuracy.

References


