Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

ADAPTIVE FUZZY NEURAL NETWORK FOR INVERSE MODELING OF ROBOT MANIPULATORS

D. T. Pham, A. A. Fahmy, and E. E. Eldukhri,

Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, U.K.

Abstract: This paper presents a new systematic adaptive fuzzy neural network for inverse modelling of robot manipulators. An inductive learning algorithm is applied to generate the required inverse modelling rules from the robot's input/output records. A full differentiable fuzzy neural network is developed to construct the inverse models of the robot manipulator, while any adaptation technique, such as back-propagation algorithm, can be applied to tune the network parameters online. *Copyright © 2008 IFAC*

Keywords: Fuzzy logic, Neural networks, Modelling, Robots manipulators, Intelligent robotics, Autotuning, Nonlinear systems, Adaptive system and control, Knowledge discovery.

1. INTRODUCTION

Designing controllers for a Multi-Input Multi-Output (MIMO) system requires clear knowledge of its mathematical model. For robot manipulators, it is almost impossible to obtain or identify precisely this model. Instead, the so called adaptive inverse model control techniques are generally used. This paper presents a systematic adaptive fuzzy neural network for inverse modelling of robot manipulators. For this purpose, an inductive learning technique (Pham, *et al.*, 2004) was modified to generate fuzzy modelling rules. A full differentiable fuzzy neural network was developed to construct the inverse model, while any adaptation mechanism can be applied to tune the network parameters online. The proposed system was tested on the virtual dynamic model of the first three links of the Puma560® robot arm. The remainder of the paper is organized as follows. Section (2) outlines fuzzy rules generation technique from input/output numerical examples. Section (3) discusses the overall structure of the proposed inductive fuzzy neural network. Section (4) presents the obtained modelling results and section (5) concludes the paper.

2. RULES GENERATION TECHNIQUE

2.1 Review.

For a dynamic system with single input u and single output y, the output at time interval k can be expressed in discrete form as in equation (1).

$$y(k) = f(y(k-1),..., y(k-n), u(k-1),..., u(k-m)) \qquad (1)$$

This equation can be extended to represent MIMO dynamic systems as well. When input/output records are used, function f, and integers n and m define the dynamic system. If n and m are given, the only task is to find function f. Fuzzy neural networks can be employed to approximate f (Narenda and Parthasarathy, 1990). Since robot manipulators can be regarded as bounded-input bounded-output (BIBO) stable systems in presence of input, equation (1) can be used to approximate their dynamics.

2.2 Inductive Learning.

A fuzzy rule used to describe a numerical record is in many ways similar to the rule created via inductive learning (Pham and Aksoy, 1995). The main difference is that, due to the notion of fuzziness, a record will have a particular "degree of match" (μ_{rule}(record)) with each rule (Srinivasan, *et al.*, 1993). The fuzzy inductive learning algorithm presented in (Pham, *et al.*, 2004) is briefly explained. The algorithm is designed to extract fuzzy "If-Then" rules from numerical records. In this paper, the output is divided into equal, 50% overlapped Gaussian membership functions (target classes). Each record (E) is described in terms of a fixed set of m (no. of inputs) attributes (equivalent to linguistic variables) and by an output class value (CE). Each created rule is composed of a number of conditions on each (or some of the) attribute(s) (CA_i) and by a class value (C_{rule}). Each rule can be represented as $CA_1 \wedge CA_2 \wedge \ldots \wedge CA_m \rightarrow C_{\text{rule}}$. In order to create a rule set, the algorithm incrementally employs a rule forming process until all records are covered. The first step is to select a seed example (SE), which is the first record in the list not covered by previously created rules. The second step consists of employing a specific search process to create a consistent and general rule covering the (SE). The main feature of this search is that the conditions for inputs are created automatically during the rule forming process. The result is a rule where all conditions will take the form [$V^i_{min} < A_i < V^i_{max}$]. These conditions might cover large areas in the records space. Thus, as the third and final step, the algorithm employs a post-processing technique that reduces the coverage of some conditions to the training data range only. The search mechanism searches for rules that cover as many records as possible from the target class and at the same time exclude records belonging to other classes. By applying this procedure, there is no need to discretise the input data. The algorithm identifies splitting points for each continuous attribute range during the learning process. Each condition takes the form [$V^i_1 < A_i < V^i_2$] where V^i_1 and V^i_2 are continuous values included in the i^th continuous attribute range [V^i_{min}, V^i_{max}]. At the end of rule forming process, transformation into fuzzy conditions is employed.

Consider the condition [$V^i_1 < A_i < V^i_2$], it is transformed into a membership function f (a, b, c), where a=V_1^i, b=V_2^i, and c= ($V_2^i + V_1^i$)/2.

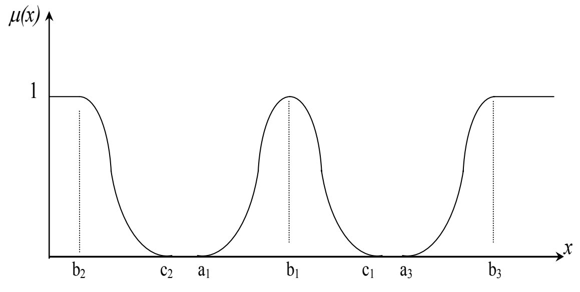

Fig. 1. Types of generated input membership functions.

If V_1^i =-∞, then a=-∞, b=V_2^i, and c=$V_{\min}^i$, which is the minimum attribute value. If V_2^i =+∞, then a=V_1^i, b=+∞, and c=$V_{\max}^i$, which is the maximum attribute value. These values are then used to generate equivalent Gaussian and sigmoid membership functions to be used in the fuzzy neural network as shown in Fig. 1. To adaptively model the inverse kinematics of the robot arm, a fuzzy rule-base is generated first using the algorithm explained. Equation (2) expresses an approximate discrete inverse kinematics relation.

$$\theta_i^{k+1} \cong f(x^{k+1}, y^{k+1}, z^{k+1}, \theta_1^k, \theta_2^k, \ldots\ldots, \theta_n^k) \tag{2}$$

Where k is the sampling interval, $i = (1,2,\ldots, n)$, n is the number of links, (x, y, z) is the end-effector's Cartesian position, and θ is the joint angle. Three sets of six-input single-output fuzzy rules can be generated representing the robot inverse kinematics. The entire training set is composed of around forty thousand records. The outputs domains have all been selected to be decomposed into eleven Gaussian membership functions. The resulting model is composed of thirty-nine rules for the first three links compared to more than eight hundred rules using Wang's method (Wang and Mendel, 1992a,b).

To adaptively model the inverse dynamics of the robot arm for inverse model control systems, a fuzzy rule-base is generated first using the algorithm explained. Equation (3) expresses an approximate discrete inverse dynamics relation for this purpose.

$$T_i^{k+1} \cong f(T_1^k, \ldots, T_n^k, \theta_1^{k+1}, \ldots, \theta_n^{k+1}, \theta_1^k, \ldots, \theta_n^k, v_1^{k+1}, \ldots, v_n^{k+1}, v_1^k, \ldots, v_n^k) \tag{3}$$

Where T is the joint torque and v is the joint velocity. Three sets of fifteen-input single-output fuzzy rules can be generated representing the robot inverse dynamics. The entire training set is composed of around forty thousand records. The outputs domains have all been selected to be decomposed into eleven Gaussian membership functions. The resulting model is composed of two hundred and thirty rules for the first three links compared to more than fourteen hundred rules generated using Wang's method (Wang and Mendel, 1992a,b).

3. INDUCTIVE FUZZY NEURAL NETWORK

3.1 Network Structure

Fig. 2 presents the structure of the proposed six-layer fuzzy neural network. The network employs time-delayed feed-backs from output layer to input layer. Using the same notation reported in (Lin and Lee, 1991), the function that provides the net input and net output of each node can be written as in equation (4).

$$Inp_{net} = f^k\begin{pmatrix} u_1^k, u_2^k, \ldots\ldots, u_p^k, \\ w_1^k, w_2^k, \ldots\ldots, w_p^k \end{pmatrix}, Out_{net} = o_i^k = a^k\left(f^k\right) \quad (4)$$

Where p is the number of inputs to the node, w is the link weight associated with each input, u is the node output in the preceding layer, k indicates the layer number, f is the node function, and $a^k(.)$ denotes the activation function in layer k.

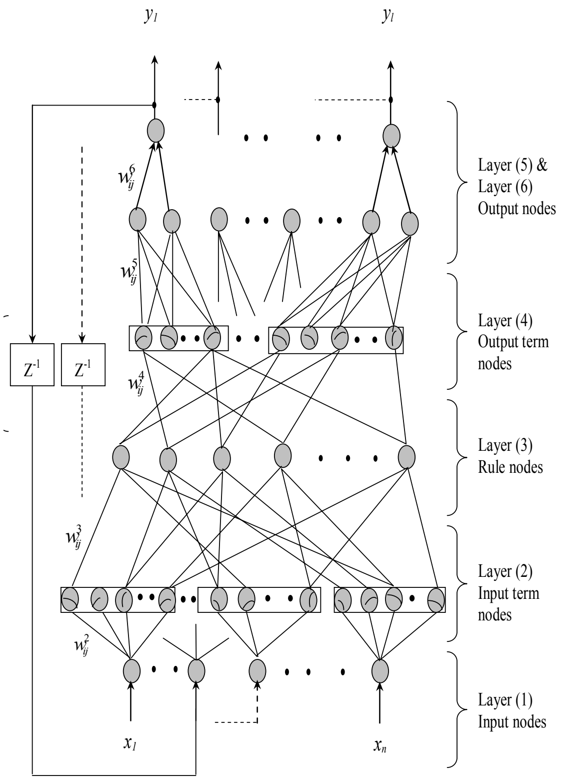

Fig. 2. Proposed fuzzy neural network.

In the proposed network in fig. 2, layer (1) contains n nodes, representing the input linguistic variables; these nodes receive a crisp input vector $X = (x_1, \ldots, x_n), f_i^1 = u_i^1 = x_i$, $a_i^1 = f_i^1$.The link weights at layer (1) are fixed to unity. Nodes at layer (2) are input term nodes which act as membership functions to represent the terms of the respective n input linguistic variables. An input linguistic variable x in a universe of discourse U is characterized by $A(x) = \{A_x^1, A_x^2, \ldots, A_x^m\}$, where $A(x)$ is the term set of x, which is the set of the generated membership functions for each input derived from inductive learning algorithm.

Layer (2) therefore accommodates n independent term sets, where each corresponds to an input x_i and is partitioned to m_i terms representing input membership functions. The function of each node j in a term set i is to calculate the degree of membership of the input x_i with respect to the membership function associated with the term set $A_j(x_i)$ according to the specific equation of this membership function as in equation (5).

$$f_{ij}^2 = \frac{\left(\left(w_{ij}^2 \times a_i^1\right) - m_{ij}\right)^2}{\sigma_{ij}^2} \quad \& \quad a_{ij}^2 = e^{-f_{ij}^2} \quad \textit{for Gaussian}$$

$$f_{ij}^2 = \frac{7\left(\left(w_{ij}^2 \times a_i^1\right) - \beta_{ij}\right)}{\left|\beta_{ij}\right|} \quad \& \quad a_{ij}^2 = \frac{1}{1 + e^{f_{ij}^2}} \quad \textit{for Left Sigmoid} \qquad (5)$$

$$f_{ij}^2 = \frac{7\left(\beta_{ij} - \left(w_{ij}^2 \times a_i^1\right)\right)}{\left|\beta_{ij}\right|} \quad \& \quad a_{ij}^2 = \frac{1}{1 + e^{f_{ij}^2}} \quad \textit{for Right Sigmoid}$$

Where m_{ij} and σ_{ij} are, respectively, the centre (or mean) and the width (or variance) of the Gaussian function and β_{ij} is the characteristic value for the sigmoid function. m_{ij}, σ_{ij}, and β_{ij} all calculated from the (a,b,c) parameters generated for each membership function from the inductive learning stage. Hence a link weight at layer (2) w_{ij}^2 can be interpreted as an adjustable free parameter of the input membership function. The tuning of this parameter (link weight) has the effect of tuning the membership function parameters (a,b,c).

The nodes at layer (3) are the inductive learning rules nodes. Hence, the rule nodes perform the *softmin* function (Berenji and Khedkar, 1992) as the minimum interpretation of the sentence connective "*and*" between the antecedents of a fuzzy rule is employed. Therefore the function of the r^{th} rule node is as in equation (6).

$$a_r^3 = f_r^3 = softmin\,(u_1^3, u_2^3, \ldots, u_q^3) = \frac{\sum_{i=1}^{q} u_i e^{-\zeta u_i}}{\sum_{i=1}^{q} e^{-\zeta u_i}} \qquad (6)$$

Where $r = 1, \ldots, R$, and R is the number of rules or rule nodes in layer (3), q is the number of inputs for that particular rule, u_i is the ith input to layer (3), and ζ is an index representing the softness of the *softmin* function. All link weights at this layer are fixed to unity to transmit only the membership degree of the linguistic input to the rule interpretation mechanism.

The nodes at layer (4) are output term nodes which act as membership functions to represent the output terms of the respective l linguistic output variables (in this case $l=3$). An output linguistic variable y in a universe of discourse W is characterized by $F(y) = \{F_y^1, F_y^2, \ldots, F_y^{11}\}$, where $F(y)$ is the term set of y, that is the set of the class membership functions for each output, as explained previously. Consequently layer (4) accommodates 3 independent term sets, where each term set corresponds to an output y_i and is partitioned to 11 terms representing output membership functions.

The nodes in layer (4) perform the *softmax* function (Estevez and Nakano, 1995). Therefore, the function of each term node j in the output term set i, is as in equation (7).

$$a^{4}_{ij} = f^{4}_{ij} = softmax(u^{4}_{1}, u^{4}_{2}, \ldots, u^{4}_{p}) = \left[1 - \frac{\sum_{i=1}^{n} \overline{u}_{i} e^{-\zeta \overline{u}_{i}}}{\sum_{i=1}^{n} e^{-\zeta \overline{u}_{i}}} \right] \qquad (7)$$

Where p is the number of rules sharing the same consequent (output term node), u_i is the ith input to layer (4), $\overline{u}_i = (1-u_i)$ is the membership logic complement, and ζ is an index representing the softness of the *softmax* function. Hence the link weights at layer four are fixed to unity. The number of nodes at layer (5) is $2l$, where l is the number of output variables, i.e. two nodes for each output variable. The function of these two nodes is to calculate the denominator and the numerator of an approximate form of Mean of Maxima (MOM) defuzzification function (Runkler, 1997) for each output variable. The functions of the numerator and denominator nodes of the i^{th} output variable are $a^{5}_{ni} = f^{5}_{ni} = a^{4}_{ij} * m_{ij}$, while $a^{5}_{di} = f^{5}_{di} = a^{4}_{ij}$. Where f^{5}_{ni} and f^{5}_{di} are respectively the node functions of the numerator and the denominator nodes of the i^{th} output variable. m_{ij} is the centre (or mean) of the Gaussian function of the j^{th} term of the i^{th} output linguistic variable y_i. Layer (5) employs $2l$ weight vectors, with two weight vectors for each output variable. The first link weight vector connects the numerator node of the i^{th} output to the term nodes in its term set and its weight components are denoted by w^{5}_{nij}. Each component of this weight vector represents the centre (or mean) of the membership function of the j^{th} term of the term set of the i^{th} output variable. The second link weight vector connects the i^{th} output denominator node to the term nodes in its term set and its weight components are denoted by w^{5}_{dij}. Hence the link weights are fixed to unity.

Nodes at layer (6) are the defuzzification nodes. The number of nodes in layer (6) equals the number of output linguistic variables. The function of the i^{th} node corresponding to the i^{th} output variable is as in equation (8).

$$y_i = a^{6}_{i} = f^{6}_{i} = \frac{w^{6}_{ni}\, a^{5}_{ni}}{w^{6}_{di}\, a^{5}_{di}} \qquad (8)$$

Where w^{6}_{ni} and w^{6}_{di} are layer (6) link weights associated with each output variable node. These two link weights represent a scaling factor of an output. Following the network construction phase, the network then enters the parameter learning phase to adjust its free parameters through on-line adaptation. The network adjustable free parameters were selected to be centre's (m^{5}_{ij}) of the output membership functions of the term nodes in layer (4) as well as the link weights at layers (2) and (6).

3.2 Parameters Tuning

The back-propagation learning algorithm is applied as an example of an adaptation mechanism which requires full differentiable model to optimally tune the proposed fuzzy neural network parameters. The problem for the supervised learning can be stated as: Given n input patterns $x_i(t)$, $i = 1,....,n$, and l desired output patterns $y_i(t)$, $i = 1,.....,l$, the fuzzy partitions, and the fuzzy rule base, adjust the network free parameters optimally. In the parameter learning phase, the network works in the feed-forward manner, that is the goal is to minimize the error function $E = \frac{1}{2}\left(y(t) - y_{net}(t)\right)^{2}$. Where $y(t)$ is the desired output, and $y_{net}(t)$ is the current network output. For each training data set, starting at the input nodes, a forward pass is followed to compute the activity levels of all the nodes in the network. Then, starting at the output nodes, a backward pass is followed to compute the rate of change of the error function with respect to the adjustable free parameters for all the hidden nodes. Assuming that (w) is the adjustable free parameter in a node, then the general learning rule can be written as $\Delta w = -\frac{\partial E}{\partial w}$ and $w(t+1) = w(t) + \eta \Delta w$. Where η is the learning rate, then using the chain rule, the partial derivative can be defined as $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial w} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial f}\frac{\partial f}{\partial w}$. Hence, the calculation of the back-propagated errors can be described starting at the output nodes. The adaptive tuning rule for the weights at layer (6) is $\frac{\partial E}{\partial w^{6}_{ni}} = \frac{a^{5}_{ni}(y_{net}(t) - y(t))}{a^{5}_{di} w^{6}_{di}}$ & $\frac{\partial E}{\partial w^{6}_{di}} = \frac{w^{6}_{ni} a^{5}_{ni}(y(t) - y_{net}(t))}{a^{5}_{di}\left(w^{6}_{di}\right)^{2}}$, while the error propagated to layer (5) is, $\delta^{6}_{ni} = \frac{w^{6}_{ni}(y_{net}(t) - y(t))}{a^{5}_{di} w^{6}_{di}}$ & $\delta^{6}_{di} = \frac{w^{6}_{ni} * a^{5}_{ni}(y(t) - y_{net}(t))}{w^{6}_{di}\left(a^{5}_{di}\right)^{2}}$.

At layer (5), no adjustment is required for the link weights connected to the denominator nodes, while an adjustment is required for the link weights w^{5}_{nij}'s which represent the centres m_{ij}'s of the output membership functions. Consequently, the adaptive rule to tune the centers of the output membership functions is $\frac{\partial E}{\partial m_{ij}} = \delta^{6}_{ni} \times a^{4}_{ij}$. The propagated error from layer (5) to the j^{th} node in the i^{th} term set in layer (4) is $\delta^{5}_{ij} = \left(\delta^{6}_{ni} \times m_{ij}\right) + \left(\delta^{6}_{di}\right)$. No adjustment is required for the link weights of layer (4). Only the error signals δ^{4}_{r}'s need to be calculated and to be propagated to a rule node r in layer (3). Each one of these error signals is a summation of L propagated error signals δ^{4}_{ri}, one error signal from a specific node *j* of each term set *i*, where *i =1,....,L* and *L* is the number of output variables. The error at layer (4) is $\delta_r^4 = \sum_i \delta_{ri}^4 = \sum_i \left(\delta_{ij}^5 \times \frac{\partial a_{ij}^4}{\partial f_{ij}^4} \times \frac{\partial f_{ij}^4}{\partial a_r^3} \right)$, $\frac{\partial a_{ij}^4}{\partial f_{ij}^4} = 1$, and

$$\frac{\partial f_{ij}^4}{\partial a_r^3} = \frac{-\left[e^{-\zeta \bar{a}_r^3}\left(1 - \zeta \bar{a}_r^3\right) \sum_{m=1}^{p} e^{-\zeta\left(\bar{u}_{ijm}^4\right)} + \zeta e^{-\zeta \bar{a}_r^3} \sum_{m=1}^{p} \bar{u}_{ijm}^4 e^{-\zeta\left(\bar{u}_{ijm}^4\right)} \right]}{\left(\sum_{m=1}^{p} e^{-\zeta\left(\bar{u}_{ijm}^4\right)} \right)^2},$$

if the *j*th term node at the *i*th term set at layer (4) is connected to the *r*th rule node at layer (3), otherwise, $\frac{\partial f_{ij}^4}{\partial a_r^3} = 0$. Where *p* is the number of rules sharing the same *j*th output node, and $\bar{u}_{ijm}^4$ is the complement of the *m*th input to the *j*th output term node at the *i*th term set at layer (4). Similar to layer (4), no adjustment is required for link weights at layer (3). Only the error signals δ_{ij}^3'*s* need to be calculated and propagated from the *r*th rule node at layer (3) to the *j*th node at the *i*th term set at layer (2). Each of these errors is a summation of *p* propagated error signals δ_{ijm}^3 from layer (2), where $m = 1, \ldots, p$, and *p* is the number of rules that share the same *j*th node at the same *i*th input at layer (2). So, $\delta_{ij}^3 = \sum_m \delta_{ijm}^3 = \sum_m \left(\delta_m^4 \times \frac{\partial a_m^3}{\partial f_m^3} \times \frac{\partial f_m^3}{\partial a_{ij}^2} \right)$, $\frac{\partial a_m^3}{\partial f_m^3} = 1$, and

$$\frac{\partial f_m^3}{\partial a_{ij}^2} = \frac{\left[e^{-\zeta a_{ij}^2}\left(1 - \zeta a_{ij}^2\right) \sum_{i=1}^{N} e^{-\zeta u_{mi}^3} + \zeta e^{-\zeta a_{ij}^2} \sum_{i=1}^{N} u_{mi}^3 e^{-\zeta u_{mi}^3} \right]}{\left(\sum_{i=1}^{N} e^{-\zeta u_{mi}^3} \right)^2},$$

if the *j*th term node at the *i*th input term set in layer (2) is connected to the rule node *m* at layer (3), otherwise $\frac{\partial f_m^3}{\partial a_{ij}^2} = 0$. Where u_{mi}^3 is the *i*th input to the rule node *m* in layer (3) and *N* is the number of input term sets. The adaptive rule to tune the weights at layer (2) is,

$\frac{\partial E}{\partial w_{ij}^2} = \delta_{ij}^3 \times \frac{\partial a_{ij}^2}{\partial f_{ij}^2} \times \frac{\partial f_{ij}^2}{\partial w_{ij}^2}$ where $\frac{\partial a_{ij}^2}{\partial f_{ij}^2} = -e^{f_{ij}^2}$ for Gaussian, $\frac{\partial a_{ij}^2}{\partial f_{ij}^2} = \frac{-e^{f_{ij}^2}}{\left(1 + e^{f_{ij}^2}\right)^2}$, for left sigmoid, and $\frac{\partial a_{ij}^2}{\partial f_{ij}^2} = \frac{-e^{f_{ij}^2}}{\left(1 + e^{f_{ij}^2}\right)^2}$ for right sigmoid functions respectively, while $\frac{\partial f_{ij}^2}{\partial w_{ij}^2} = \frac{2a_i^1\left(\left(a_i^1 w_{ij}^2\right) - m_{ij}\right)}{\sigma_{ij}^2}$, for Gaussian, $\frac{\partial f_{ij}^2}{\partial w_{ij}^2} = \frac{7a_i^1}{\left|\beta_{ij}\right|}$ for left sigmoid, and $\frac{\partial f_{ij}^2}{\partial w_{ij}^2} = \frac{-7a_i^1}{\left|\beta_{ij}\right|}$ for right sigmoid functions respectively. The propagated error from layer (2) to layer (1) is $\delta_i^2 = \sum_j \delta_{ij}^2 = \sum_j \left[\delta_{ij}^3 \times \frac{\partial a_{ij}^2}{\partial f_{ij}^2} \times \frac{\partial f_{ij}^2}{\partial a_i^1} \right]$, while $\frac{\partial f_{ij}^2}{\partial a_i^1} = \frac{2 \times w_{ij}^2\left(\left(w_{ij}^2 \times a_i^1\right) - m_{ij}\right)}{\sigma_{ij}^2}$ for Gaussian, $\frac{\partial f_{ij}^2}{\partial a_i^1} = \frac{7 \times w_{ij}^2}{\left|\beta_{ij}\right|}$ for left sigmoid, and $\frac{\partial f_{ij}^2}{\partial a_i^1} = \frac{-7 \times w_{ij}^2}{\left|\beta_{ij}\right|}$ for right sigmoid functions respectively and $\frac{\partial a_{ij}^2}{\partial f_{ij}^2}$ is calculated as before. The nodes in this layer transmit input values to the next layer directly without any processing. So, the link weights at layer (1) are fixed to unity and no tuning is required in this layer. Following the construction phase and the learning phase, an on-line tuning process is performed to obtain the optimum mapping for the inverse kinematics and inverse dynamics of the robot manipulator. The network adjustable free parameters were selected to be centres ($m_{ij}s$) of the output membership functions of the term nodes in layer (4) as well as the link weights at layers (2) and (6) as mentioned earlier. According to the above explanation, any adaptation mechanism can be applied to the proposed fuzzy neural network due to its full differentiable characteristics.

4. RESULTS

The proposed method was tested on the first three links of the Puma 560® robot. Fig. 3 to Fig. 5 represents the pre-adaptation results for the inverse kinematics modeling for random trajectories compared with the actual values. Fig. 6 to Fig. 8 represents the pre-adaptation results for inverse dynamics modeling normalized result to rated torque for random trajectories compared with the actual outputs. It can be seen from the modeling results that the suggested modeling method is effective.

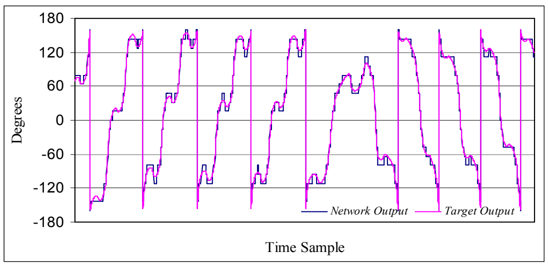

Fig. 3. Results for link-1 angle prediction.

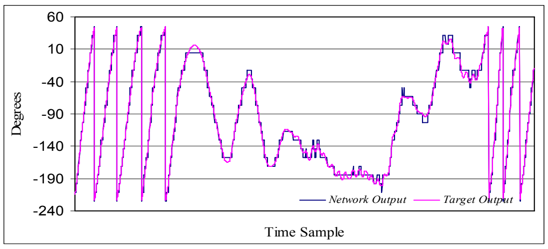

Fig. 4. Results for link-2 angle prediction.

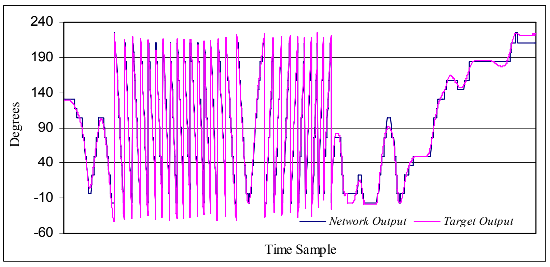

Fig. 5. Results for link-3 angle prediction.

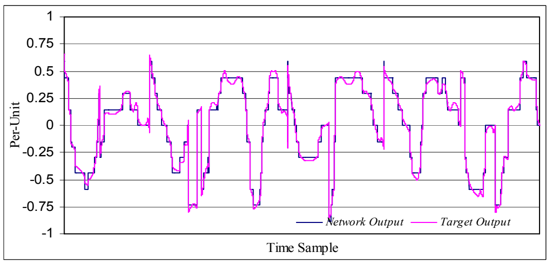

Fig. 6. Results for link-1 torque prediction.

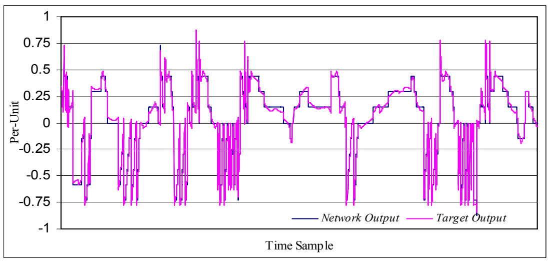

Fig. 7. Results for link-2 torque prediction.

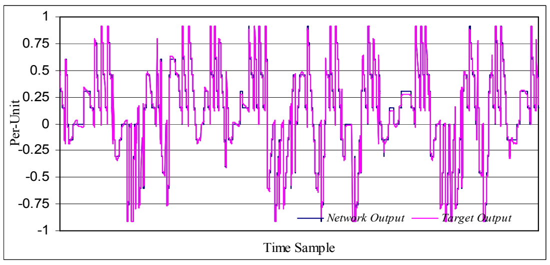

Fig. 8. Results for link-3 torque prediction.

5. CONCLUSION

This paper proposed a method for inverse modeling of robotic manipulators for inverse-model based control system. The main idea is the use of inductive learning technique to develop fuzzy rule sets to mimic the models from numerical records. These rule sets were employed in a full differentiable fuzzy neural network to adaptively tune the obtained models in adaptive inverse control systems. The method was tested using the Puma 560® robot model. Results showed that the method was successful and applicable for inverse modeling of robot arms.

ACKNOWLEDGEMENT

The research described in this paper was supported by the EC FP6 Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

REFERENCES

Berenji, H. R. and P. Khedkar (1992), Learning and Tuning Fuzzy Logic Controllers Through Reinforcements, IEEE Transactions on Neural Networks, volume 3, issue 5.

Estevez, P. A. and R. Nakano (1995), Hierarchical Mixture of Experts and Max-Min Propagation Neural Networks, IEEE International Conference on Neural Networks, Australia.

Lin, C. T. and C. S. G. Lee. (1991), Neural Network-Based Fuzzy Logic Control and Decision System, IEEE Transactions on Computers, volume 40, issue 12.

Narendra, K. S. and K. Parthasarathy (1990), Identification and Control of Dynamical Systems Using Neural Networks, IEEE Transactions on Neural Networks, volume 1, issue 1.

Pham, D. T., S. Bigot and S. S Dimov. (2004), A new technique for fuzzy rule induction, Proceedings of the fourth CIRP International Seminar on Intelligent Computation in Manufacturing Engineering, Sorrento.

Pham, D. T. and M. S. Aksoy (1995), A New Algorithm for Inductive Learning, Journal of Systems Engineering, volume 5.

Runkler, T. A. (1997), Selection of Appropriate Defuzzification Methods Using Application Specific Properties, IEEE Transactions on Fuzzy Systems, volume 5, issue 1.

Srinivasan, A., C. Batur, and C.C. Chan (1993), Using Inductive Learning to Determine Fuzzy Rules for Dynamic Systems, Engineering Applications of Artificial Intelligence, volume 6, issue 3.

Wang, L. X. and J. M. Mendel (1992a), Fuzzy basis functions, universal approximation, and orthogonal least-squares learning, IEEE Transactions on Neural Networks, volume 3, issue 5.

Wang, L. X. and J. M. Mendel (1992b), Generating Fuzzy Rules by Learning from Examples, IEEE Transactions on Systems, Manufacturing, and Cybernetics, volume 22, issue 6.