

# Parallel Algorithm for Feedforward Neural Network Controller

Z.Hanzálek

Trnka Laboratory for Automatic Control  
Department of Control Engineering  
Karlovo n. 13, 121 35 Prague 2  
Czech Technical University in Prague, Czech Republic  
hanzalek@rttime.felk.cvut.cz

**Abstract.** A problem with simulation of multilayer neural network on transputer array is described in this article. The decomposition and mapping on given architecture is proposed as well as a simple message passing scheme. Practical experiments with inverted pendulum plant are described including useful hints for setting neural network architecture and tuning neural networks parameters. Then neural controller is designed and criterion function is defined for given control problem. Finally the real-time aspects of the controller design are outlined.

## 1 Introduction

When designing real-time controller it is necessary to deal with application demands (a plant complexity, time constants, reliability) and algorithm requirements (time complexity, memory space). In real time digital control the controller output must be performed within the loop sample interval on the basis of periodically sampled measurements. The choice of the controller architecture is a way how to satisfy the requirements coming from the nature of the real-time control. Possible way is to use a parallel processing system composed of several processing elements which can operate concurrently, communicating with each other when necessary.

## 2 Parallel Algorithm

### 2.1 Neural network algorithm

The following equations specify a function of the stochastic gradient learning algorithm, where  $N_l$  is the number of neurons in layer  $l$ ,  $k$  denotes an algorithm iteration number,  $I_j^l(k)$  denotes input to the cell body of neuron  $j$  in layer  $l$ ,  $u_j^l(k)$  denotes output of neuron  $j$  in layer  $l$ ,  $\delta_i^l(k)$  denotes the error back propagated through the cell body of neuron  $i$  in layer  $l$ ,  $w_{ij}^l(k)$  denotes weight of synapse between cell body  $i$  in layer  $l-1$  and cell body  $j$  in layer  $l$ ,  $\eta^l$  denotes the learning rate, and  $\alpha^l$  denotes the momentum term in layer  $l$ .

Activation - Forward Propagation

$$I_j^l(k) = \sum_{i=1}^{N_{l-1}} [w_{ij}^l(k) \cdot u_i^{l-1}(k)] \quad (1)$$

$$u_j^l(k) = f(I_j^l(k)) = \frac{2}{1 + e^{-I_j^l(k)/T}} - 1 \quad (2)$$

Error Back Propagation - Output layer

$$\delta_i^l(k) = f'(I_i^l(k)) \cdot (u_i^{desired}(k) - u_i^l(k)) \quad (3)$$

Error Back Propagation - Hidden layer

$$\delta_i^l(k) = f'(I_i^l(k)) \cdot \sum_{j=1}^{N_{l+1}} (\delta_j^{l+1}(k) \cdot w_{ij}^{l+1}) \quad (4)$$

Learning - Gradient Method

$$\Delta w_{ij}^l(k) = \eta^l \cdot \delta_j^l(k) \cdot u_i^{l-1}(k) + \alpha^l \cdot \Delta w_{ij}^l(k-1) \quad (5)$$

$$w_{ij}^l(k) = w_{ij}^l(k-1) + \Delta w_{ij}^l(k) \quad (6)$$

## 2.2 Communication requirements

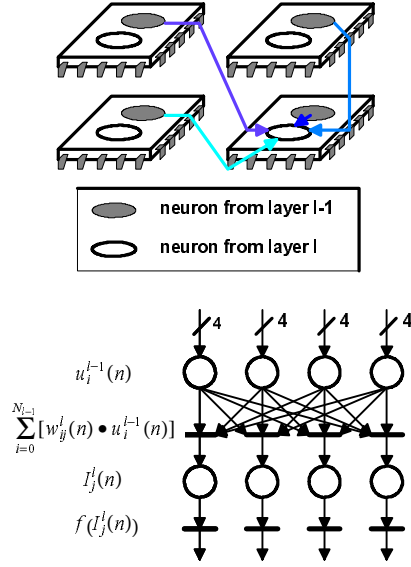
One part of activation procedure in hidden layer having four neurons is represented in figure 1 . Let us assume that this activation step is calculated in four processors. Then each neuron at each processor has to receive a previous layer output from all other nodes.

It is evident that an efficient algorithm could be realized only in a case when we minimize the amount of data communicated among processors. The basic idea is to split the neuron into synapses (in sequential activation represented by equations ( 1 ) , ( 4 ) , ( 5 ) and ( 6 )) and a cell body (corresponding to the sigmoid function - equations ( 2 ) , ( 3 ) and ( 4 )). The splitting operation makes it possible to simulate each neuron by different processors and minimize communication.

## 2.3 A Toroidal Lattice Architecture

The problem decomposition is done by the network of virtual processors (VPs). The VPs could be divided into the three categories:

- synapse processor ( SP )
- cell processor ( CP )
- input/output processor IO which is present only once. His behaviour is more complex, once acts as synapse, once as cell and once performs communication with a host computer.



**Figure 1.** The part of activation in four processors and its Petri net representation

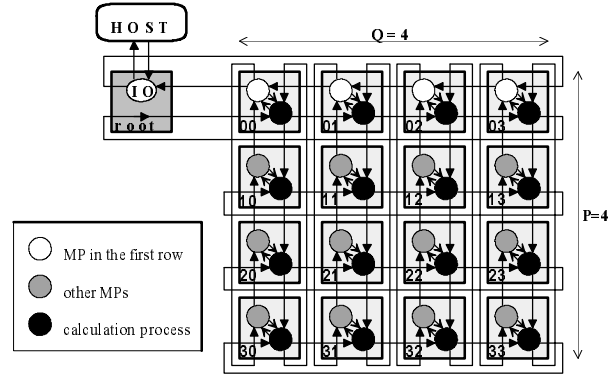
VPs having four communication channels are arranged in the toroidal lattice architecture (Fujimoto, 1992 [2]; Murre, 1993 [5]). Each SP performs operations corresponding to the functions of a synapse in neural network that are the sum operation and updates its weight. The CP has the functions corresponding to the activation sigmoid function and error evaluations. All SPs and CPs are connected to their four neighbours by unidirectional channels (up, down, left, right).

## 2.4 Data distribution

Training data delivering is the crucial problem for efficient parallel processing in the simulation of large scale neural networks. The training data in typical application are available on one processor - typically on a root processor connected to host computer. If we assume that neurons from the input layer are "placed" on a root transputer (respective sigmoid function of input layer could be calculated in advance) then input data are delivered to the first row of NPs. Output layer neurons are present in all NPs so the output has to be send from all NPs to the root and the output error has to be delivered from the root to all NPs.

If we want to run the described algorithm on an array of processors connected by links we have to deal with the data distribution problem. The solution proposed in figure 2 is to create communication process MP on each physical

processor and to connect this MP to process performing calculation. MPs in the first row are connected with root in ring as well as all MPs in each column. All MPs are connected by the same physical links as calculation processes but in opposite direction. All data among communication processes are delivered with the identification number. Each MP is assigned as a high priority process.



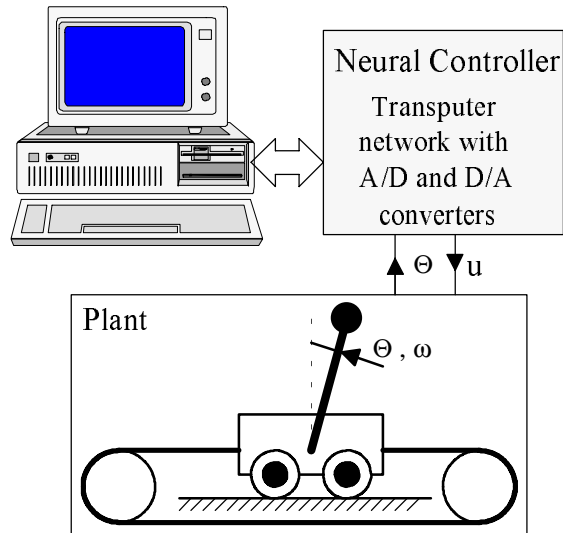
**Figure 2.** Realization on array of 17 transputers

For more details on implementation and experimental results see [3].

### 3 Neural Controller

Besides the performance measurement the neural network controller was implemented. Experiments were done on an inverted pendulum physical model LIP manufactured by the Amira company. This problem has been studied by many control theorists and engineers as an inverted pendulum problem. The task is to stabilize the pendulum in upright position having information about pendulum angle  $\theta$  and angular speed  $\omega$ . Based on the measurements the controller generates a suitable signal  $u$  which controls the DC motor. The controller was realized by the transputer network with ADT108 and DAT208 converters in a form of TRAM modules. Such a system allows to realise fast controller with sampling frequency up to 1kHz (this limit is given by the communication with A/D and D/A converters). Figure 3 shows schematically the experimental environment.

The inverted pendulum is a non-linear system that is linearized by approximation of goniometric functions at the operating point. Plant input is limited by the saturation of the control voltage for the motor ( $\pm 10V$ ). The most significant nonlinearities acting on the cart are the dry friction and the static friction.



**Figure 3.** Neural controller with inverted pendulum plant

### 3.1 Copying an existing controller

At first conventional PD controller was designed by the second Ziegler-Nichols method (refer to Ogata, 1990 [6]). Such a discrete controller given by the equation (7) is very easy to tune by finding critical gain  $K_{cr}$  and critical period  $P_{cr}$ . These values were found by setting  $K_d = 0$  and rising up  $K_p$  from 0 to the critical one.

$$u(k) = K_p \cdot e(k) + K_d \cdot [e(k) - e(k-1)]/T_v \quad (7)$$

Where  $e(k) = \Theta(k) - \Theta_{desired}(k)$  is the input to the controller and  $T_v$  is the sampling period (in our experiments  $T_v = 10ms$ ).

The closed loop system was stable and the conventional linear PD controller with non-linear friction compensator served us for comparison with a neural controller.

In order to gain experience in setting neural network architecture and tuning neural network parameters a simple task 1 was examined - off line learning of the controller function using a random generator to generate the input data. Then simple task 2 was performed - on line copying the existing conventional controller with a neural network.

Doing this and studying reference Hush and Horne (1993) [4] validity of following facts was tested:

- From the algorithm convergence purposes the momentum is limited by  $0 < \alpha < 1$ .

- The weights are initialized to small random number ( refer to the back-propagation phase).
- The use of the linear output nodes tends to make learning easier.
- The learning rate  $\eta$  is inversely proportional to the gain of the network - so it is useful to normalize input and output signals, then  $\alpha + \eta \approx 1$ .
- The neural network with one hidden layer will be sufficient for our purpose, using networks with more layers leads to the problems with propagating error to the input layer. According to the theorem of Weierstrass and to the study of de Villier (1992) [11] more layers do not seem to be necessary.
- The upper bound on the number of hidden layer neurons is generally a function of the number of training samples. Concerning to the simplicity of controller functions in our case not more than 8 neurons were used.
- Learning NN with 4 inputs  $[\theta(k), \theta_{desired}(k), \theta(k-1), \theta_{desired}(k-1)]$  was easy and fast (10 seconds) in the task 1 because input data generated by the random generator covered all input range.
- Learning NN with 4 inputs  $[\theta(k), \theta_{desired}(k), \theta(k-1), \theta_{desired}(k-1)]$  was impossible in the task 2 because input data did not cover all input range, namely in the case of the desired values.
- Learning neural network with 2 inputs  $[e(k), e(k-1)]$  was possible and relatively fast in the task 2.

The profit of this approach lies in the method consisting of the two following phases:

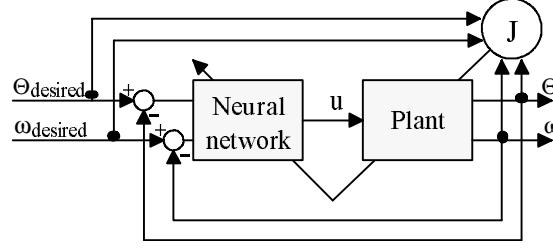
- 1) Neural network was taught to function in the same way as the non optimal controller able to stabilize a plant - weights pre-setting.
- 2) In order to optimise the controller behaviour a new criterion function was defined and the neural controller was applied on line with the plant to adapt its parameters - weights tuning.

The fundamental problem formulated by Barto (1991) [1] is: where does the training information come from ? In a control application target plant output is known but not target network output - control signal. This problem appears when phase 2 is performed or when the neural controller starting from the random values on line with plant is realized. Corresponding questions and solutions are clarified in the following paragraph.

### 3.2 Learning control architecture

Figure 4 shows the feedback controller, implemented as neural network, with its output  $u$  driving the plant (in the terms of equations specifying neural network algorithm  $u = u_1^{output}$ ). Specified architecture using neural network as controller require training information, that is based on the knowledge of measured values and the criterion function. This procedure requires knowledge of the Jacobian of the plant.

Our learning algorithm for the multilayer neural network uses a stochastic gradient search technique to find the network weights that minimize a criterion



**Figure 4.** Neural controller learning architecture

function. The criterion function to be minimized at each time step is the weighted sum of squared errors:

$$J = \frac{1}{2} [c_{\Theta}(\Theta(k) - \Theta_{desired}(k))^2 + c_{\omega}(\omega(k) - \omega_{desired}(k))^2] \quad (8)$$

Error back propagation cannot be applied directly to the special learning architecture because of the location of the plant. Referring to figure 4, the plant can be thought of as an additional, although unmodifiable, layer (refer to Psaltis, 1988 [7]). Then the total error is propagated back through the plant using the partial derivatives of the plant at its operating point. So equations specifying back-propagation phase have to be recalculated in order to minimize  $J$ .

$$\frac{\partial J}{\partial w_{ij}^l} = \frac{\partial J}{\partial I_j^l} \frac{\partial I_j^l}{\partial w_{ij}^l} = \frac{\partial J}{\partial I_j^l} u_i^{l-1} \quad (9)$$

There is no change for the input and hidden layers, so for the output layer consisting of one linear neuron and for  $\Theta_{desired} = \omega_{desired} = 0$  the chaining rule is used, once more:

$$\frac{\partial J}{\partial I_{out}} = \frac{\partial J}{\partial u} \frac{\partial u}{\partial I_{out}} = \left( \frac{\partial J}{\partial \Theta} \frac{\partial \Theta}{\partial u} + \frac{\partial J}{\partial \omega} \frac{\partial \omega}{\partial u} \right) \cdot 1 \quad (10)$$

In practice the neural controller was used in order to avoid the plant identification, so only qualitative knowledge of the plant should be used. This means that the Jacobian of the plant will be approximated by +1 or -1 :

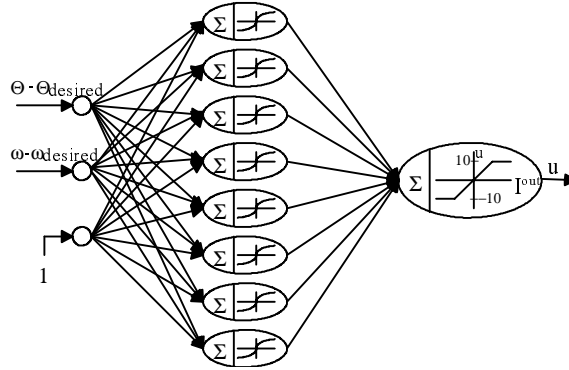
$$\frac{\partial J}{\partial I_{out}} = \frac{\partial J}{\partial \Theta} \cdot 1 + \frac{\partial J}{\partial \omega} \cdot 1 = 2 \cdot \frac{(c_{\Theta}\Theta)}{2} + 2 \cdot \frac{(c_{\omega}\omega)}{2} \quad (11)$$

So the equation ( 3) changes in the case of the on-line neural controller with  $I_{out} \in (-10, +10)$  to the form:

$$\delta^{out}(k) = c_{\Theta}\Theta(k) + c_{\omega}\omega(k) \quad (12)$$

Figure 5 shows the neural controller under consideration. It is evident that for  $I_{out} \notin (-10, +10)$  the equation changes to the form:  $\delta^{out} = 0$ .

Practically experimenting with the neural controller it was found out that bias input to the output layer causes many problems in the learning procedure. The reason lies in the fact that control loop closes over  $\delta^{out}$  and not over controller input. In such cases the system works but changes its weights all the time.



**Figure 5.** Internal structure of neural controller

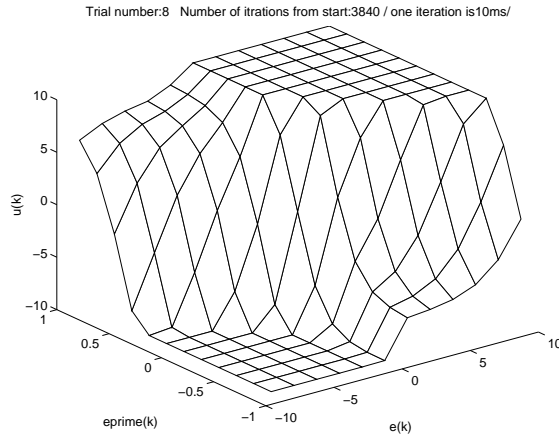
Using the architecture given in the figure 5 with the input vector  $[e(k), e_{prime}(k)]$  better results were achieved than using the second controller with the input vector  $[e(k), e(k - 1)]$ . Referring to the equation (7) there should not be a big difference, but the second controller was not able to distinguish the angular speed  $\omega$ . This problem is connected with the value of the sampling frequency. In the other words the second controller functioned more like P one than like PD one.

### 3.3 Experimental results

Applying neural controller on the inverted pendulum we have performed several experiments similar to ones published by Barto *et al.*(1983) [1], Saerens and Soquet (1991) [8] and others. The controller was able to stabilize the plant (at least for 15 minutes) after less than 10 trials. At the beginning of each trial the pendulum was put to the origin  $\Theta = 0$ . The controller was considered that failed when  $\Theta \notin (-10, +10)$ . The number of trials was dependent on the initial weights generated by the random generator and on the disturbances as well. In order to have similar conditions for all experiments the initial weights were set to small random numbers so that  $u$  did not reach saturation and the learning procedure could take place. The weights usually converged in 10 seconds ( $T_v = 10ms$ ).

Monitoring a neural controller shape it was seen that the system behaved in a way very similar to man's thinking. First dependency on  $\Theta$  was chosen ( P-controller), then dependency on  $\omega$  was adjusted ( D-controller).

Assuming the learning procedure to be already stopped the neural controller could be seen as a non-linear static function with two inputs and one output as shown in the figure 6. Its behaviour is very similar to the conventional linear PD controller with saturation. In order to compare the stability of both controllers an investigation of the non-linear controller stability has to be made, which could be done for example by the approximation of  $u$  by its first harmonic.



**Figure 6.** Neural controller characteristics

The evolution of the neural controller at the time is dependent on neural network architecture and parameters, but the final shape of the neural controller is given namely by the ratio  $c_\Theta/c_\omega$  defined by the criterion function  $J$ . This ratio seems to be the function of the plant parameters, namely critical period  $P_{cr}$ . In the other words - target controller behaviour is influenced by defining the criterion function  $J$ , but this behaviour is not known when we have no quantitative knowledge of the plant, so it is very difficult to set  $J$  when the plant is seen as a black box.

### 3.4 Real-time aspects

Designing real-time controller in the OCCAM environment we have to deal with several questions coming from the nature of real-time control. Typically we need at least two processes running in parallel and communicating each other - process control and process monitor.

The process control performs all control functions in limited time given by the sampling period. The process monitor communicates with the operator (display, keyboard, disk) that means there is no safety that these functions could be done in a defined time.

## 4 Conclusion

This paper presents a toroidal lattice architecture used to simulate multilayer neural network. Application of neural controller to drive physical plant was examined and described. The same system could be used in a case of plants with higher demands because using parallel architecture, we can gain from the advantages of parallel processing (increased computational speed, easy expansion within a uniform hardware and software, closer relationship between the inherent parallelism expressed at the design stage, flexibility etc.)

## Acknowledgements

This research has been conducted at *Trnka Laboratory for Automatic Control* (supported by the Ministry of Education of the Czech Republic under VS97/034).

## References

1. Barto, A.G.,1991, *In: Neural Networks for Control* (Miller, W.T., Sutton, R.S., Werbos, P.J, Ed.) Chap.1 - Connectionist Learning for Control, MIT Press, pp.5-58.
2. Fujimoto, Y., N.,Fukuda, T., Akabane,1992, Massively Parallel Architectures for Large Scale Neural Networks Simulations, *IEEE Transactions on Neural Networks*, **3**, No. 6, pp.876 - 887.
3. Z. Hanzálek, A Parallel Algorithm for Gradient Training of Feedforward Neural Networks, accepted for publication by *Parallel Computing*, Elsevier Science.
4. Hush,D.R., B.G., Horne,1993, Progress in Supervised Neural Networks, *IEEE Signal Processing Magazine*, January 1993, pp.8-39.
5. Murre,J.,1993) Transputers and Neural Networks: An Analysis of Implementation Constrains and Performance, *IEEE Transactions on Neural Networks*, **4**, No. 2, pp.284 - 292.
6. Ogata, K.,1990, *Modern Control Engineering*, Prentice-Hall International 1990, pp. 597-604.
7. Psaltis, D., A., Sideris, A.A., Yamamura,1988, A Multilayered Neural Network Controller Based on Back-Propagation Algorithm, *IEEE Control Systems Magazine*, April 1988, pp.17-21.
8. Saerens, M., A., Soquet,1991, Neural Controller based on back-propagation algorithm, *IEE Proceedings-F*, **138**, No. 1, pp.55-62.
9. Sutton, R., A.G., Barto, C., Anderson,1983, Neuron-like Adaptive Elements that can solve Difficult Learning Control Problems, *IEEE Trans.*, SMC-13, **5**, pp.834-846.
10. Tollenaere, T., G.A., Orban,1991, Simulating modular neural networks on message-passing multiprocessors, *Parallel Computing*, No. 17, North-Holland, pp. 361-379.
11. de Villier,J., E., Bernard,1992, Backpropagation Neural Nets with One and Two Hidden Layers, *IEEE Transactions on Neural Networks*, **4**, No.1, January 1992, pp.136-141.