The locally linear Nested Network for robot manipulation

P. van der Smagt and F. Groen and F. van het Groenewoud

Abstract— **We present a method for accurate representation of high-dimensional unknown functions from random samples drawn from its input space. The method builds representations of the function by recursively splitting the input space in smaller subspaces, while in each of these subspaces a linear approximation is computed. The representations of the function at all levels (i.e., depths in the tree) are retained during the learning process, such that a good generalisation is available as well as more accurate representations in some subareas. Therefore, fast and accurate learning are combined in this method.**

The method, which is applied to hand-eye coordination of a robot arm, is shown to be superior to other neural networks.

I. Introduction

In our research, we concentrate on the pick-and-place task in the kinematic domain. The system provides information about the angles of the robot's joints (measured through joint angle sensors) and a pixel array available from a single camera mounted in the robot's end-effector. This placement of the camera increases visual precision when the robot's hand is near the target and avoids problems of occlusion and pixel correspondence. The position of the target, as seen by the camera, together with the joint angle information of the robot is used to generate a joint angle rotation with which the robot must reach the target.

Traditional robot control is based on precise models of the sensors, the robot, and the environment which together constitute the control problem. Using these models, typical tasks such as pick-and-place and object avoidance can be completely analysed and solved. However, when robots intervene with changing environments, static models are not sufficient anymore, but must cope with adaptations of the robot (such as wear and tear) and its sensors (e.g., calibration and re-calibration). Also, unavoidable errors due to discrepancies between the model and the machine must be corrected.

Neural adaptive robot controllers that have been previously proposed either suffer from long learning times or from a less precise approximation. For example, Kohonen networks as initially proposed by Ritter *et al.* [3, 4] can get a precision of around 0.5 cm in the end-effector position in a few feedback steps, but need thousands of iterations to attain reasonable results. The use of a single feed-forward network trained with conjugate gradient back-propagation has been shown to give fast and highly adaptive approximation of inverse kinematics functions, but a large number of feedback steps is needed to get high-precision results [9, 7]. Differently put, local representation methods attain high precision but need many learning samples (which are expensive to get for a real robot system), whereas global approximations can quickly establish reasonable overall approximations but the final result will be coarse; also, parallel implementation is more cumbersome.

In previous work, we have combined global and local representations in a method called the *nested network method* [8, 2]. The approximation starts with a global map in the form of a fast-learning feed-forward network, but as learning proceeds local maps will be built in a tree-like structure where needed. This method has been shown to be very economic in its use of learning samples; however, the training of a large number (approximately 100) of feed-forward networks consumes considerable computing power. In this paper, we further investigate the method by use of local linear approximation instead of nonlinear feed-forward approximators; learning cost is then independent of the amount of information already gathered.

Control structure The method presented in this paper is applied to learning the hand–eye coordination for a simulated six degrees of freedom (DoF) anthropomorphic robot arm called the OSCAR–6 robot; see figure 1. The reach space used in the ex-

The authors are with the Department of Computer Systems, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands, fax +31–20–525-7490, phone +31–20–525-7524, email smagt@fwi.uva.nl

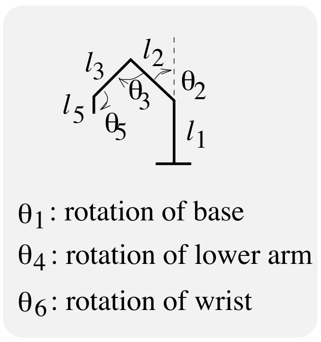
l_2
l_3
θ_3
θ_2
l_5
θ_5
l_1
θ_1 : rotation of base
θ_4 : rotation of lower arm
θ_6 : rotation of wrist

Figure 1: Structure of the OSCAR–6 robot.

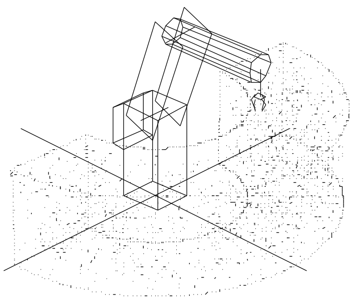

Figure 2: Reach space of the OSCAR–6 robot.

periments is shown in figure 2. By keeping the camera, which is mounted in the end-effector of the robot, always looking downwards, we have restricted ourselves to a 3 DoF system which suffices for many industrial applications (e.g., performing pick-and-place operations). Also, it places such restrictions on the image processing software that object position identification can be realised in real time (i.e., object identification within the time needed for grabbing one image frame). Since visual observation is ego-centered, we have chosen to control the robot with delta joint values, i.e., rotations from the current state of the robot.

The aim of our system is to get the object with known dimensions in the centre of the camera image at a predefined size. Given a current state of the robot $\vec{\theta} = (\theta_2, \theta_3)$ and the object position $\vec{v} = (x, y, A)$ where A is the observed area of the object, extracted from the camera, the network must generate delta joint values $\vec{\vartheta} = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$ to reach the object position. Note that the value of the θ_1 is not necessary as network input. From the joint and camera information input samples to train the control system can be constructed using an input-adjustment method [5].

II. The Nested Network Method

In this section we will highlight the principal components of the nested network method; see also [2].

Methods used for continuous function approximation usually allow their user to specify a 'smoothness' parameter. Such a parameter is determined based on *a priori* knowledge of the function at hand, while overfitting due to noise must be prevented. Especially for one-dimensional fitting problems, such approximation methods are very advanced and successful.

However, such methods never scale to approximation of functions of high dimensionality. It is in this field that 'black-box' approximations with neural networks have proven their worth.

However, it is important that the approximator realises a more 'smooth' function where the gradient of the data does not vary too much, while being flexible in other parts. In the nested network method, we realise this by maintaining multi-resolution representations of the data. At the lowest resolution, a single hyperplane is set to 'approximate' the data so as to realise a minimal summed squared error. At the highest resolution, the data points themselves are available; on intermediate levels, linear interpolations of the data are created for subspaces of the entire input space.

Suppose the method is used for approximation of a function of n variables. Initially, the approximation is done only by the root node. Incoming learning samples are assigned to one of its 2^n subspaces, which are obtained by dividing the input space for this node in two halves along each dimension. Using incremental update, the linear approximator is adapted by each learning sample assigned to it. When, however, in one of its 2^n subspaces, the average error in approximation exceeds a threshold, a new node is created which is subsequently representative for this subspace.

This process of splitting up will be repeated until for every part of the input space the desired approximation precision is achieved. This process can be compared with the first part of the well-known *split and merge* process as used in image processing, as first mentioned in [1].

A second requirement is that we want to have a very flexible system, able to adjust itself to changes in the system. This means that from the moment that the representation as constructed in the tree does no longer match the measured system (i.e., the learning samples are no longer represented by the existing feed-forward networks), the tree has to be chopped down. This can be achieved by a merge process. During this process learning samples from the 'old' environment have to be thrown away together with the corresponding linear interpolators. In fact, the learning process has to be restarted, depending on how much the system has changed.

Figure 3 shows the network nodes of a partly constructed nested network over a two-dimensional input space.

In our nested network approach, the input space of an n-dimensional function that has to be approximated, can be recursively divided in 2^n subspaces until the desired fragmentation is attained for the storage of the input samples. This is done by halving the subspace in each of the n dimensions knowing for each dimension its minimum and maximum value. Now any of the subspaces can now be found in at most d steps with d the recursion depth (the recursion depth is equal to the maximum depth of the tree). We restrict the immediate subspaces of a node to be of equal size for reasons of implementational facility only.

A. Tree creation

The system builds an internal representation of the high-dimensional tree. A node in the tree represents a subspace of the input space U.

Index. Let us denote an *index* as a natural number such that

$$0 \leq \text{index} < 2^{\text{inputdim}} \tag{1}$$

where 'inputdim' denotes the number of dimensions of the input space. In our case, index ranges from 0 to 31 inclusive. Each node in the tree can have at most 32 children, each representing an equally large subspace of U. A child of a node is uniquely identified by its index.

Bin. Learning patterns are stored in *bins*. Each bin is uniquely defined by an array of indices i_1, i_2, ..., i_{maxdepth}, where 'maxdepth' denotes the maximum depth the tree can reach. This way, a bin represents a small subspace of the total input space U. Learning patterns will be stored in the bin that can represent them. If a bin does not yet exist for a particular learning pattern, it will be created. Hence, no more bins than learning patterns can exist.

Node. The tree is built up out of *nodes*. A node is able to approximate the subspace it represents. A node can be uniquely identified by its depth and an array of indices. It has a pointer to its parent node (except for the root node) and up to 32 pointers to its children.

Split. A node is split when the error the node produces is higher then a user defined threshold value. The error of a node is defined as follows:

$$E_{\text{node}} = \sum_{i=0}^{\text{patterns}} E_{\text{pattern},i} \tag{2}$$

while the error of a single pattern is the summed squared one, i.e., the Euclidian distance between the desired and the actual end-effector position.

Each node has a linked list of bins attached to it. Only bins that represent a part of the input space that the node represents are linked to that node. Initially, all bins will be linked to the root node. At that point, the root node represents the entire input space U. This is a coarse approximation and the root node will soon be split because the error it produces is relatively high. Before that, all bins are linked to the root node and subsequently all learning patterns are stored in those bins. As soon as a split occurs for one of the 32 subspaces of a node, a new approximator is created. Then the parent node is searched for bins that fall the new subspace of the U. Those bins are unlinked from the parent node and linked to the new child of the node that was split. Then the node is learned with these patterns.

III. Simulation results

For the feed-forward networks presented in the simulations, networks with five hidden units each were chosen. The networks are trained with conjugate gradient optimisation [6].

The maximum depth that the tree is allowed to grow should, theoretically, be infinity. However, due to the finiteness of computer memory, a value is chosen higher than the maximum depth the tree (i.e., the deepest network node) would reach in a typical run. In our case, we set the maximum depth to six, which proves to be sufficient for our purposes.

A. Motor babbling

In order to give the nested network the chance to learn the function, an exploratory phase is required. After all, if the network starts with random weights, it is likely that it will always generate very similar robot commands, which are subsequently reinforced since they constitute new learning patterns: the input space will not be explored. A solution of this problem is to perform some random movements, and train the network with the corresponding learning samples. It will now be able to generate movements in different directions and the input space can be explored.

The robot has to reach its target position within a precision of 1 mm. A maximum of 20 feedback steps is allowed. While the robot is moving, camera and joint encoder values are measured, and stored as learning samples in the tree. Figure 4 shows the

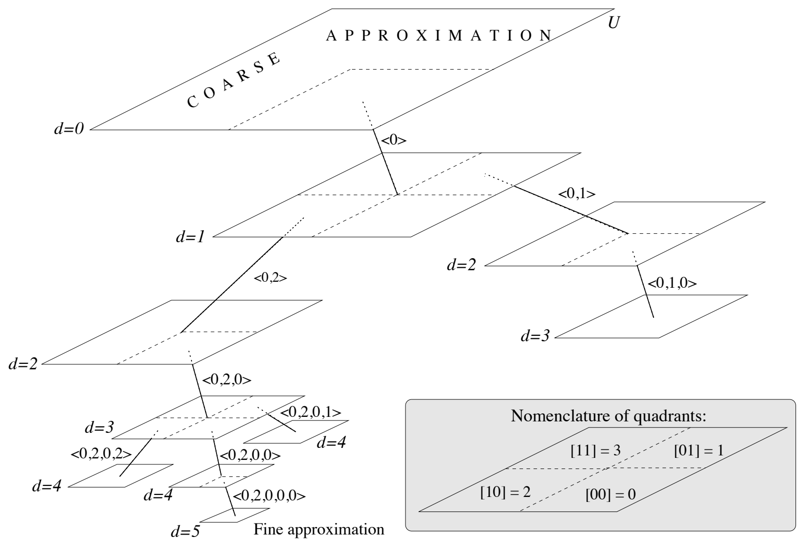

Figure 3: Example of the subdividing of a two-dimensional space U. Each space can be divided into four equally sized parts. If no further subdividing exists the best possible approximation for that specific subspace is reached.

grasping error in cm. when the robot is allowed to make three (two feedback steps) moves. The most notable feature is the attained speedup compared to the feed-forward network implementation, while keeping the precision at the same level.

We compare the grasping error attainable with the nested network with those attained with a Kohonen and a single feed-forward network. A Kohonen network builds up a local representation of the reach space of the robot. Each neuron in the network has a small receptive field, in which a Jacobian is responsible for a locally linear representation. Thus, the attainable accuracy should not be worse than that obtained with our method. From our simulations we found that the network reaches a precision of around 0.5 cm with two steps towards the target, but needs many learning steps to get this result 6.

Conversely, a single feed-forward network builds up a global nonlinear approximation by a summation of sigmoids. It is therefore expected that a coarse approximation is very quickly attained (using a conjugate gradient algorithm), but the final accuracy will be not so good. This is shown from the simulations: the feed-forward network approach exhibits fast learning but will get a lower accuracy (figure 7). Note the different scale in this figure.

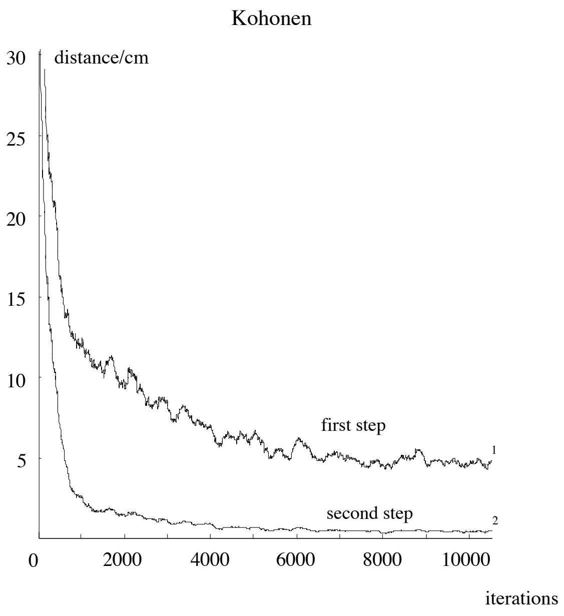

Figure 6: The grasping error for the robot trained with a Kohonen network [2]. This curve has been smoothed with a moving average filter of width 20.

IV. Conclusion

We presented a method which can represent unknown multi-dimensional functions from samples randomly drawn from its input space. The system creates a multi-resolution representation of the input space using locally linear approximators. By using linear approximations, the method is computationally very cheap but still gives very accurate results.

When compared with feed-forward networks and

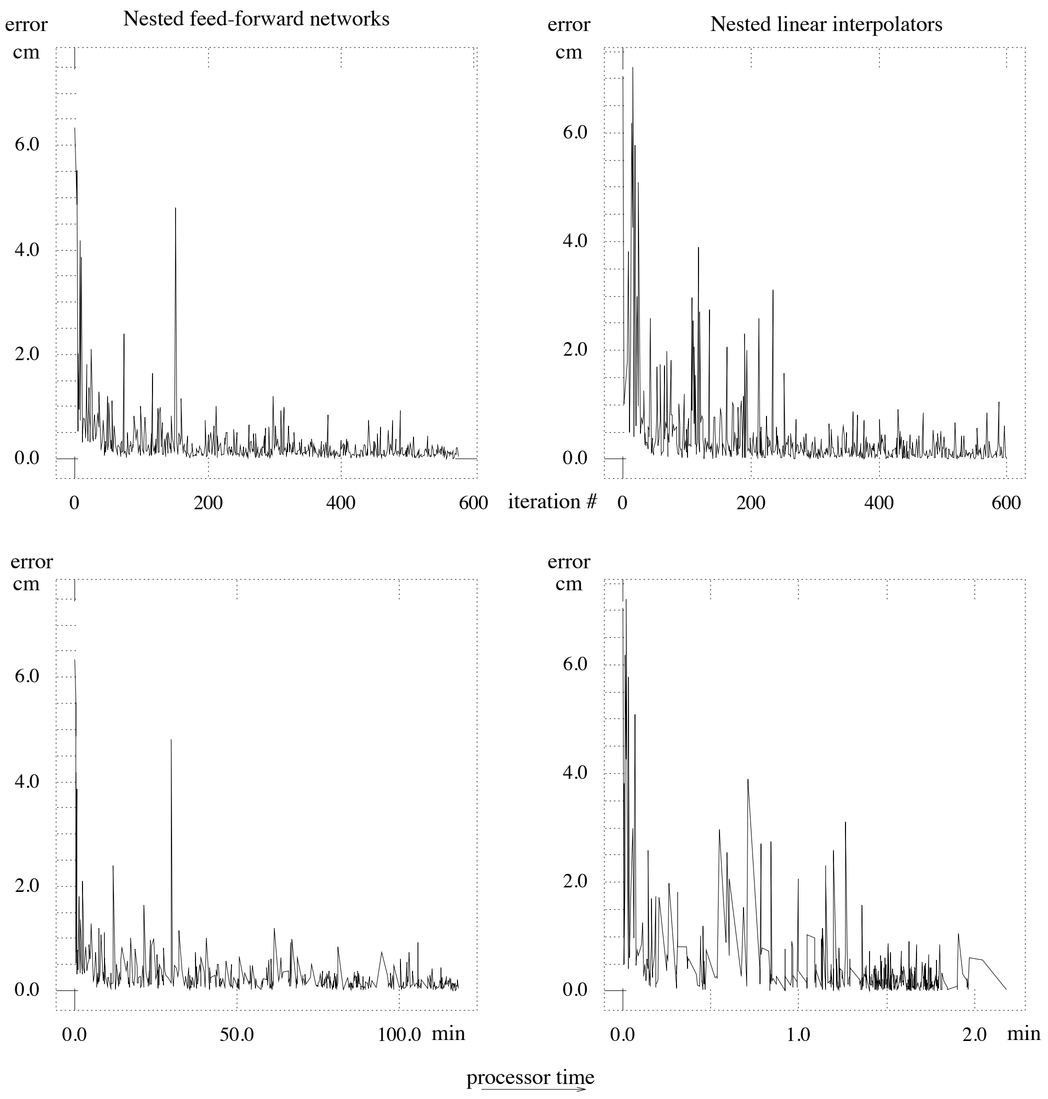

Figure 4: Results of the linear interpolation method in comparison with the earlier presented results. All graphs show the distance in cm. between the end-effector and the object after three steps (i.e., two feedback steps). The nested feed-forward network and the nested linear network perform similarly well when compared on the iteration number (top row). However, when compared on the processor time needed for building the tree and learning the nodes (bottom row), the linear method is superior, being around 60 times as fast for the first 500 learning iterations.

Kohonen networks for the same task, this method of high-dimensional function approximation is shown to be superior in its economic use of learning samples, i.e., it generalises faster while also allowing very exact representation.

References

[1] C. Brice and C. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1:205–226, 1970.

[2] A. Jansen, P. van der Smagt, and F. C. A. Groen. Nested networks for robot control. In A. F. Murray, editor, *Neural Network Applications*. Kluwer Academic Publishers, 1993. In print.

[3] T. Martinetz, H. Ritter, and K. Schulten. Learning of visuomotor-coordination of a robot arm with redundant degrees of freedom. In R. Eckmiller, G. Hartmann, and G. Hauske, editors,

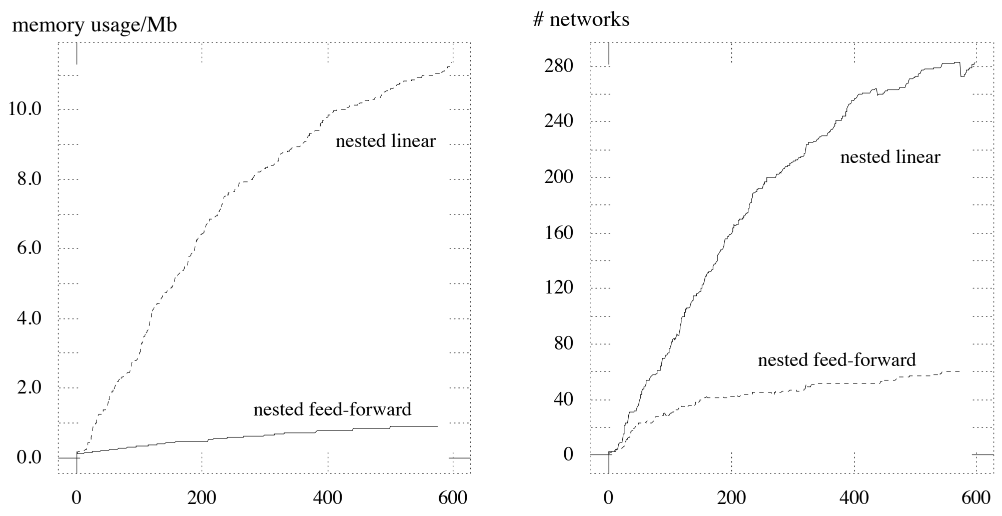

Figure 5: Memory usage (left) and number of nodes (right) for the nested feed-forward network method and the nested linear perceptron method.

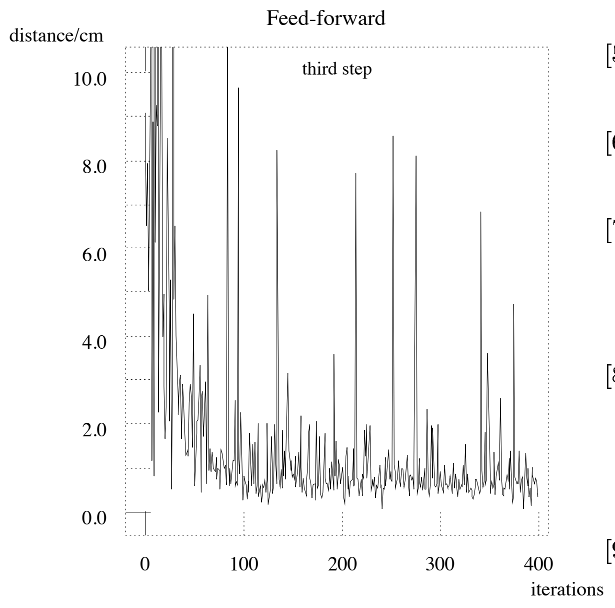

Figure 7: The grasping error for the robot trained with a single feed-forward neural network with 27 hidden units steps [7].

Parallel Processing in Neural Systems and Computers, pages 431–434. Elsevier Science Publishers B.V., 1990.

[4] T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, volume 1, pages 397–402, Espoo, Finland, 1991.

[5] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayer neural network controller. *IEEE Control Systems Magazine*, 8(2):17–21, April 1988.

[6] P. van der Smagt. Minimisation methods for training feed-forward networks. *Neural Networks*, 7, January 1994.

[7] P. van der Smagt, F. Groen, and B. Kröse. Robot hand-eye coordination using neural networks. Technical Report TR CS–93–10, Department of Computer Systems, University of Amsterdam, Amsterdam, September 1993.

[8] P. van der Smagt, A. Jansen, and F. C. A. Groen. Interpolative robot control with the nested network approach. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 475–480, Glasgow, Scotland, U.K., 11–13 August 1992.

[9] P. van der Smagt and B. J. A. Kröse. A real-time learning neural robot controller. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, pages 351–356. Elsevier Science Publishers, Espoo, Finland, June 1991.