БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, 48
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, 48

София . 1999 . Sofia

Application of Recurrent Neural Networks for Systems Identification and Control*

Ieroham Baruch, Alexander Mitev, Bojka Nenkova

Institute of Information Technologies, 1113, Sofia

1. Introduction

The Neural Network (**NN**) modelling starts with the pioneering work of McCulloch and Pitts, cited in [1]. The main difference between **NN** and the computer consists of the massive parallelism and the automatic context information processing [1]. Different authors postulate different applications of the artificial **NN**, generalized as [1, 2, 3, 4]:

– Approximation of functions;

– Association used by the associative (context-addressable) memory, accessed by the context;

– Pattern recognition (classification, clustering and categorization). Well-known applications of pattern classification include character recognition, speech recognition, **EEG** waveform classification, blood cell classification and printed circuit board inspection. Well-known clustering applications include data base, data compression and data analysis;

– Process (time series) prediction;

– Identification and control, [1]. More advanced forms of neurocontrol are discussed in [1].

The diversity of these tasks prove the universality of **NN**s as an information-processing system. All these tasks are problems of learning and mapping from possibly noisy examples. Without the imposition of prior knowledge, each of the tasks is ill-posed in the sense of nonuniqueness of possible solution mappings. In practice **NN**s need to be integrated into a consistent system engineering approach. Specifically, a complex problem of interest would be decomposed into a number of relatively simple tasks that **NN**s are assigned to perform.

There are sources which show the relations between the architecture of the NNs and other features like training paradigms, learning rules, learning algorithms and the performed tasks, [2]. The **NN** architecture can be examined from different points of view, the most important of them being as it follows:

– Type of the input: binary or continuous,[3];

– Feedback presence or absence, [1, 2]. According to it **NN**s can be divided into

* This paper is partially supported byNational Science Foundation, grant No 611/ 96.

feedforward and recurrent (feedback) **RNN**s;

– Types of the connectionist network [1]. According to it the **NN**s can be: feedforward networks with a single layer and feedforward multilayer networks with partial or total connectivity between the neurones. The neurones that do not communicate with the environment are known as hidden neurones.; **NN**s with delay units and with or without hidden neurones; lattices. A lattice network is a feedforward network with the output neurones arranged in rows and columns of different dimensions.

Other important characteristics of **NN**s are topology and learning. Classification of the **NN**s according to this characteristics, is given in Appendix 1, [2].

The aim of this paper is to familiarize the readers with some of the works done in the department of Parallel Information Systems (**PIS**) from the Institute of Information Technologies (**IIT**), Bulgarian Academy of Sciences (**BAS)**, in the field of **NN** theory and applications.

1. RNN for systems identification and process prediction

A new two-layered **RNN** architecture, named Recurrent Trainable **NN** (**RTNN**), appropriate for nonlinear dynamic systems identification, was developed , [5, 6]. A state space representation of both continuous and discrete time mathematical models of **RTNN** is given in two layer Jordan Canonical Architecture (**JCA**), and a new improved **BP** type learning method, is proposed. Some topology improvements aimed to preserve **RTNN** stability and to enhance **RTNN** architecture using the saturation instead of a sigmoid function, are suggested. The proposed **RTNN** model is linear in small and nonlinear in large, which permits to apply all well known state- and output linear systems design methods like pole assignment and quadratic cost optimal control. Simulation results of nonlinear systems identification by **RTNN** and an improved **BP** learning, are given.

The main problem of the **NN** systems identification and the neural control systems design is the lack of universality, because different authors used different types of **NN** according to its application. The duality of the problem of systems identification and one step process prediction, on one side and the optimal quadratic cost control - on the other side, shows that both problems could be solved using the **RTNN** approach, defined by Baruch et all. [5, 6]. Thus, the design of **RTNN**, as an universal tool for systems identification and control, is a new trend in the adaptive control systems design.

1.1. Description of the RTNN-JCA

B a r u c h et al. [5, 6], for the first time describe **RNN** in an universal way, using the state-space approach. They defined a global linearized **RNN** model (the **RTNN**) model and studied its stability by means of the first stability law of Liapunov. The first improvement is dedicated to preserve **RNN** model stability during learning. The improved **RTNN** model is aimed to identify nonlinear dynamic processes. Two types of processes are suggested: processes, nonlinear on their output and processes, nonlinear on their state (bilinear). The **RTNN** architecture easily solves also the problem of the one-step ahead process prediction.

Let us consider the mathematical state-space description of both the Jordan continuous and discrete-time two-layer **RTNN** models, given in [5] in the form:

$$\boldsymbol{v} = J\boldsymbol{v} + B\boldsymbol{u},\ \boldsymbol{w} = S(\boldsymbol{v}),\ \boldsymbol{y} = C\boldsymbol{w},\ \boldsymbol{z} = S(\boldsymbol{y}) \tag{1.1}$$

$$\boldsymbol{V}(k+1) = J\boldsymbol{V}(k) + B\boldsymbol{U}(k),\ \boldsymbol{W}(k) = S[\boldsymbol{V}(k)],$$

$$\boldsymbol{Y}(k) = C\boldsymbol{W}(k),\ \boldsymbol{Z}(k) = S[\boldsymbol{Y}(k)] \tag{1.2}$$

where: $\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{u}, (\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{U})$ are respectively n-, l-, m - vectors, considered as **RTNN** continuous and discrete-time models state, output and input; $\boldsymbol{y}, \boldsymbol{v}, (\boldsymbol{Y}, \boldsymbol{V})$ are l-, m - vectors, respectively; J = block-

diag (J_i), B, C are constant matrices with compatible dimensions, considered as the weight matrices of the **RTNN** continuous and discrete-time models; k is a discrete - time integer variable; $\boldsymbol{S}(x)$ is a vector valued sigmoid function, i.e.:

$$\boldsymbol{S}'(x) = [s(x_1),\ s(x_2), ..., s(x_j)], \tag{1.3}$$

$$s(\text{inp}) = 1/[1+\exp(-\text{inp})],\ \text{inp}=S(d_i x_i + d_{io}) \tag{1.4}$$

where: inp is the input of the sigmoid function; d_i, d_{io} are trainable constant weights of the **RTNN**; $\boldsymbol{S}'(\boldsymbol{x})$ significates a vector transpose of $\boldsymbol{S}(\boldsymbol{x})$.

The continuous and discrete-time version of the **RTNN** mathematical model is sufficient from dynamical point of view, because if the linearized model is stable – as the function $s(x_i)$, given by (1.4), is a single decreasing and bounded – then the nonlinear models (1.1) and (1.2) will also be stable according to the first stability law of Liapunov.

The main advantages of the proposed two layer **RTNN-JCA**, defined by (1.1), (1.2) are:

a) It is described in state-space form (**SISO** or **MIMO**) and could serve as an one-step ahead state predictor/estimator.

b) The **RTNN** model is nonlinear in large and linear in small, so the matrices J, B, C obtained as a result of learning could be used for analytical design of linear state/output control laws. By means of a similar transformation the **JCF** could be transformed into Luenberger's Canonical Form which is easy to use for pole assignment design of control systems. The matrices J, B could be used for an optimal control systems design with quadratic performance index. The matrices $\boldsymbol{J}, \boldsymbol{B}, \boldsymbol{C}$ also could be used for an optimal **P, PI**, **PID** control systems design. Finally, the matrices $\boldsymbol{J}, \boldsymbol{B}, \boldsymbol{C}$ could be used in an adaptive iterative square-root algorithm for optimal control with quadratic cost criterion.

c) The **RTNN** could solve the optimal control problem itself by means of **NN** mapping.

The **RTNN** two-layer architecture contains hidden and output layers. The output layer is a **BPNN** and the hidden layer is a recurrent **JCA NN.** It was assumed that each Jordan block of it has only (1x1) or (2x2) dimension. The continuous **RTNN** model will be stable iff the system eigenvalues have negative real parts. The discrete-time **NN** model will be stable iff system eigenvalues are inside the unit circle. Then the analysis of the **RTNN** model controllability, observability and identifiability becomes easy. The last concept, taken from systems theory, gives us the possibility to check if the obtained global **RNN** model could be learned or not, [5]. From the block structure of B and C', corresponding to the block structure of J, it is possible to conclude that iff the input matrix $\boldsymbol{B}$ has zero blocks – the **RTNN** model is uncontrollable and iff the transpose of the output matrix $\boldsymbol{C}$ has zero blocks – then the **RTNN** model is unobservable. If one of both occurs the **RTNN** model is unidentifiable, which means that the **RTNN** model is untrainable. To preserve the **RTNN** stability during the training, it is necessary to impose some restrictions on the model feedback, introducing a sigmoid vector function in it, which changes the eqns. (1.1) and (1.2) in the forms:

$$\boldsymbol{v} = \boldsymbol{q} + B\boldsymbol{u},\ \boldsymbol{q} = S(J\boldsymbol{v}),\ \boldsymbol{w} = S(\boldsymbol{v}),\ \boldsymbol{y} = C\boldsymbol{w},\ \boldsymbol{z} = S(\boldsymbol{y}), \tag{1.5}$$

$$\boldsymbol{V}(k+1) = \boldsymbol{Q}(k) + B\boldsymbol{U}(k),\ \boldsymbol{Q}(k) = S[J\boldsymbol{V}(k)],\ \boldsymbol{W}(k) = S[\boldsymbol{V}(k)],$$

$$\boldsymbol{Y}(k) = C\boldsymbol{W}(k),\ \boldsymbol{Z}(k) = S[\boldsymbol{Y}(k)]. \tag{1.6}$$

Another improvement of the **RTNN** architecture is to facilitate its realisation, approximating the sigmoid function $s(\text{inp})$ with a saturation:

$$\text{sat}(\text{inp}) = \begin{cases} +1, & \text{inp} \geq +1 \\ \text{inp}, & 0 \leq \text{inp} < +1 \\ 0, & \text{inp} < 0. \end{cases} \tag{1.7}$$

Both improvements of the **RTNN** architecture are tested by simulation examples during the learning.

1.2. RTNN learning

Simultaneously with the **RNN** topology improvements, some advanced researches have been done on the methods of **RNN** learning, which naturally depends on the **RNN** topology. Baruch et all. [5] in their previous works, defined a new Jordan **RTNN** architecture with both exponential and oscillatory dynamics, which feedback weights are trainable. Some work has been done to improve this learning algorithm introducing modern refinement techniques like momentum rule, weight fixing and pruning. The most common used **BP** updating rule, applied for the two layer **RTNN -JCA**, [5] , is the following:

$$\boldsymbol{D}_{ij}(k+1)=\boldsymbol{D}_{ij}(k) + \eta \Delta \boldsymbol{D}_{ij}(k), \tag{1.8}$$

where: D_{ij} is the ij-th weight element of each weight matrix in the **RTNN** model to be updated; ΔD_{ij} is the weight correction of D_{ij}; η is the learning rate parameter. The **RTNN** model weight matrices here are denoted by $\boldsymbol{D}$ for the sake of generality. The weight corrections of the updated matrices in the discrete-time **RTNN** model, described by eqn. (1.6), are given as follows:

– For the output layer:

$$\Delta \boldsymbol{C}_{ij}(k) = \boldsymbol{[T}_j(k) - \boldsymbol{Z}_j(k)\boldsymbol{]}\, \boldsymbol{Z}_j(k)\,[1 - \boldsymbol{Z}_j(k)]\, \boldsymbol{W}_i(k) \tag{1.9}$$

where: $\Delta \boldsymbol{C}_{ij}$ is the weight correction of the $\boldsymbol{ij}$-th elements of the $(l \times n)$ learned matrix $\boldsymbol{C}$; $\boldsymbol{T}_j$ is a j-th element of the target vector; $\boldsymbol{Z}_j$ is a j-th element of the output vector; $\boldsymbol{W}_i$ is an i-th element of the input vector of the output layer, i.e. the hidden layer output.

– For the hidden layer:

$$\Delta \boldsymbol{B}_{ij}(k) = \boldsymbol{R}\, \boldsymbol{U}_i(k), \tag{1.10}$$

$$\Delta \boldsymbol{J}_{ij}(k) = \boldsymbol{R}\, \boldsymbol{V}_i(k-1), \tag{1.11}$$

$$\boldsymbol{R} = \boldsymbol{C}_i(k)\,[\boldsymbol{T}(k) - \boldsymbol{Z}(k)]\, \boldsymbol{W}_j(k)\,[1 - \boldsymbol{W}_j(k)], \tag{1.12}$$

where: $\Delta \boldsymbol{B}_{ij}$ is the weight correction of the $\boldsymbol{ij}$-th elements of the $(m \times n)$ learned matrix $\boldsymbol{B}$; $\boldsymbol{C}_i$ is a row vector of dimension $(1 \times l)$, taken from the transposed matrix $\boldsymbol{C}'$; $\boldsymbol{[T-Z]}$ is a $(l \times 1)$ output error vector, through which the error is backpropagated to the hidden layer; $\boldsymbol{U}_i$ is an $\boldsymbol{i}$-th element of the input vector $\boldsymbol{U}$; $\boldsymbol{V}_i$ is an i-th element of the vector $\boldsymbol{V}$; $\Delta \boldsymbol{J}_{ij}$ is the weight correction of the ij-th elements of the l $(n \times n)$ block-diagonal matrix $\boldsymbol{J}$ under learning; $\boldsymbol{R}$ is an auxiliar matrix with compatible dimensions. The matrix elements of 0 and 1 values will not be updated. The same equation for **RTNN** learning may be applied for the continuous-time case, eqn. (1.5).

An improvement of the **BP** updating algorithm (1.8) is to introduce a momentum term, proportional to the past $(k-1)$-th weight correction, as it is:

$$\boldsymbol{D}_{ij}(k+1) = \boldsymbol{D}_{ij}(k) + \eta\, \Delta \boldsymbol{D}_{ij}(k) + \alpha\, \Delta \boldsymbol{D}_{ij}(k-1), \tag{1.13}$$

where: α is a momentum learning rate parameter.

This correction is appropriate to perform in the case when significant error-function oscillations occur. A lot of experiments of learning with different rates of learning h and a has been done. The experiments show that the optimal combination of these learning parameters is obtained when the following inequality condition yields:

$$\boldsymbol{r}_{max} < \text{sqrt}\,(\eta^2 + \alpha^2) < 1; \quad \boldsymbol{r}_{max} = \max |\, \lambda_i | \tag{1.14}$$

where λ_i is the maximum eigenvalue of the object.

Another improvement of the **RTNN** learning algorithm, successfully applied for the **BP** learning of discrete-time **RTNN**s consider unimportant units pruning and non-useful connections removing [8]. Both methods remove the units or the weights, whose outputs or values tend to

be zero. Simultaneously with the nodes pruning, a weight fixing could be applied. During the intensive experiments done, it was observed that some of the hidden neurones change their weights rather slow with respect to the others, i.e. their weights tend to reach constant values. If the corresponding weight correction tends to zero during one epoch of learning, the learning algorithm fixes the corresponding weight or node and interrupts the process of learning. This operation reduces the total learning time almost twice. Both learning algorithms, performing weights pruning and fixing lead to exclusion of weights or nodes from the process of learning. There are two possibilities: to fix some weights or to fix the whole node. The first is more efficient then the second because in this case we do not need to compute the node error.

1.3. Simulation experiments

The improved learning algorithm for **RTNN** was tested with several linear and nonlinear dynamic objects. The topology improvements are also carefully studied. The chosen epoch size is of 1000 cycles. The learning process is finished when both the error of learning and the error of testing are reduced to an error threshold of about 1.5%. The number of epochs to reach this prescribed error is called time of learning. The input learning signal of the object and the **RTNN** consists of mixed random series of sinusoidal and rectangular patterns with a random frequency and a random amplitude, which ensures that the **RTNN** will be learnt by a wide-band variable-spectrum input signal. The quality of the **RTNN** is verified also by a test signal of an one-epoch length, given in the form:

$$\boldsymbol{u}(k) = \begin{cases} \sin(\pi k/25), & 0 < k < 251, \\ 1.0, & 250 < k < 501, \\ -1.0 & 500 < k < 751, \\ 0.3\sin(\pi k/25) + 0.1\sin(\pi k/32), & \\ +\,0.6\sin(\pi k/10), & k < 1001. \end{cases} \tag{1.15}$$

The state-space equations of both examples are given below:

$$\begin{aligned} \boldsymbol{h}(k) &= 0.3\boldsymbol{u}(k)[1 + \boldsymbol{x}_1(k) - \boldsymbol{x}_2^2(k)], \\ \boldsymbol{x}_1(k+1) &= \boldsymbol{x}_2(k), \\ \boldsymbol{x}_2(k+1) &= -0.15\,\boldsymbol{x}_1(k) + 0.8\boldsymbol{x}_2(k) + \boldsymbol{h}(k), \\ \boldsymbol{y}(k+1) &= \boldsymbol{x}_1(k+1) - 0.2\boldsymbol{x}_1^2(k+1) + 0.1\boldsymbol{x}_1^3(k+1) + 0.5. \end{aligned} \tag{1.16}$$

Two **SISO** nonlinear discrete-time simulation examples are considered. The first one is a bilinear object with a sigmoid. The second one is a nonlinear object with a saturation.

Object 1: **SISO** bilinear object (sigmoid):

The simulation results are shown in the figures. Each figure has the following common items:

(a) *Output of* ***RTNN*** *(dashed line) and* ***Object*** *(solid line) during training*

(b) ***RTNN*** *error of testing phase (dashed line) and training (solid line) at each epoch*

(c) *Output of* ***RTNN*** *(dashed line) and* ***Object*** *(solid line) during last testing phase*

Two different cases of η=0.7; α=0 and η=0.5; α=0.5 are considered for each object. As it can be seen from the results of Fig. 1, 2, 3, 4 the introduction of momentum term in the **BP** learning algorithm smoothens the error (LER is the learning error and TER is the testing error).

The case of changing the sigmoid function with a saturation for both objects augments the error of learning in the beginning but the error decreased to the same value in the final phase. The oscillations of the error of learning also augmented. The weight fixing decreases the time of learning with 30%. The learning time is 20 epochs.

Object 2: **SISO** bilinear object (saturation)

The time of learning is 10 epochs. The error of learning reached for both objects is below 1.5%.

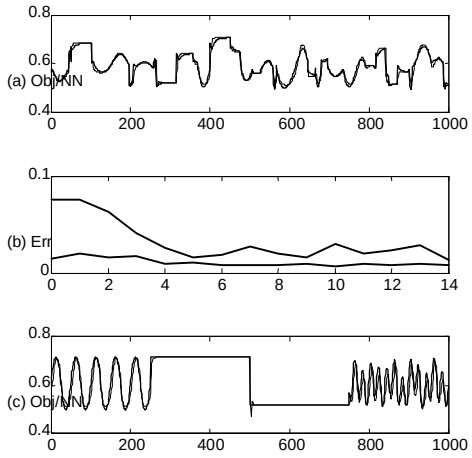

Fig.1. h=0.7, α=0.,sigm., LER=1%,TER=1,5%

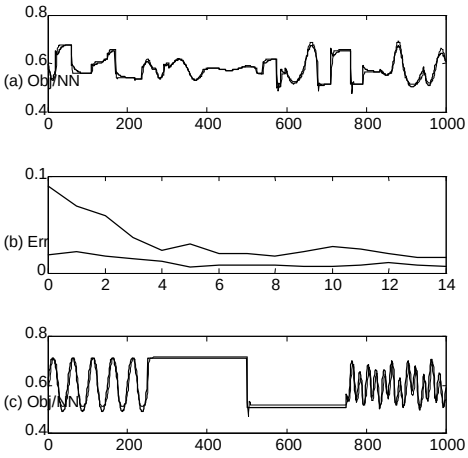

Fig.2. h=0.5, α=0.5, sigm., LER=1.5%, TER=1%

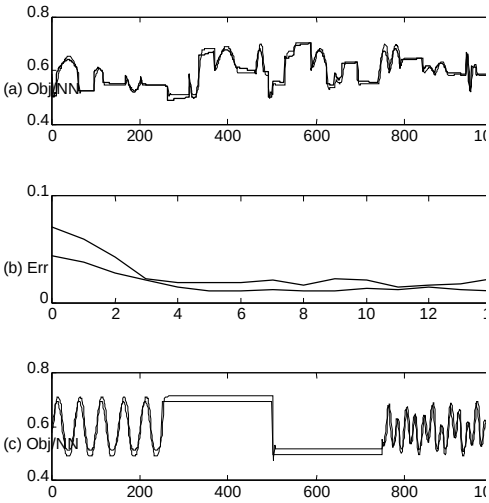

Fig.3. h=0.7, α=0, sat., LER=1,5%, TER=1,5%

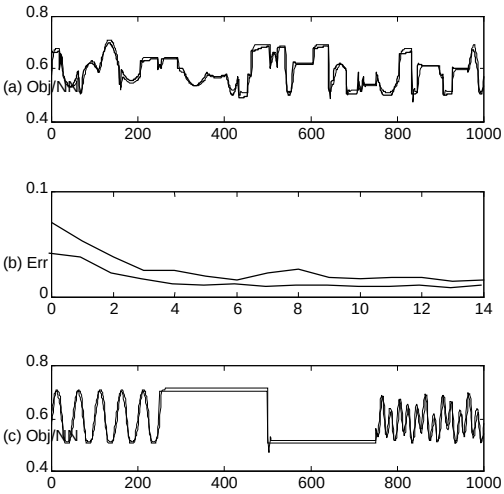

Fig. 4. h=0.5, α=0.5, sat., LER=1.5%, TER=1.25%

2. NN for approximation of functions and kimematic robot control

A kinematic control system architecture for redundant robots-manipulators, avoiding obstavles, based on context sensitive hierarchical **NN** approach, is proposed. The **NN** control system structure contains two **NN**'s. the first one is a Functional **RNN**, transforming the vector of the variation of Cartesian position into the vector of variation of the joint position, by means of the Jacobean pseudoinverse function (**JPF**). The second one is a Context three-layer **BPNN**, trained to approximate the **JPF** by means of the **BP** learning. The **BP** learning is realised to optimise robot path resolving two main problems: singular configurations and existence of multiple solutions. Method efficiency and **NN** kinematic control possibilities are demonstrated by appropriate example. This work is a collaborative work of the **CINVESTAV**, **IPN**, MEXICO, **IIT**, **BAS**, **PIS** dept.[7].

2.1. Robot kinematics description

During the last decade many works were carried out on the solution of the Inverse Kinematic Problem (**IKP**) for Robot Manipulators (**RM**), cited in [7]. The methods proposed for the general case can be classified into two main groups: particular and general. These methods are based on two models: geometric and kinematic , based on the equations:

(2.1) $$\boldsymbol{x} = \boldsymbol{F}(\boldsymbol{x}),$$ Geometric Model,

(2.2) $$\boldsymbol{x} = \boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{q}$$, Kinematic Model,

where $\boldsymbol{x}$ is a 6-dimensional vector of the Cartesian position; $\boldsymbol{q}$ is a n-dimensional join position vector; $\boldsymbol{F}(x)$ is a nonlinear vector function; $\boldsymbol{J(q)}$ is a $(n \times n)$ Jacobian matrix.

The particular method for kinematic control is based on the direct solution of the nonlinear trigonometric vector equation (4.1). This method is applied for particular cases of robots-manipulators and gives explicit analytical solution, which is its principle disadvantage. The general methods for kinematic control based on the kinematic equation of motion (4.2) give solution for every type of robot (redundant or not). These methods are independent of geometric and kinematics structure of manipulators and perform a desired optimisation criterion with null space solutions. The general approach has been used for obtaining $\boldsymbol{q}'$.

(2.3) $$\boldsymbol{q} = \boldsymbol{J}^{+}\boldsymbol{x} + (\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J})\,\boldsymbol{y},$$

where $\boldsymbol{J}^{+}$ is the pseudoinverse matrix of the Jacobean $\boldsymbol{J}$; $\boldsymbol{y}$ is an arbitrary nonzeroed n-dimensional vector.

This method of kinematic control has the following disadvantages: computationally expensive; numerically unstable due to error accumulation; not free from arm singularities (the inverse become discontinued). From this reason it is important to apply the **NN** architecture to provide highly parallel computations to obtain the solution of (2.3). There exists a variety of problems, concerning the **NN** (2.3) **AF**, as follows:

a) this function describes **discontinuous** - singular configuration; b**)** it gives an **infinite number of solutions**, because of the robot redundancy. The approach [7] proposes a continuous min error solution of the eqn. (2.3), overcoming the mentioned above problems. It is:

(2.4) $$\| \boldsymbol{x} - \boldsymbol{J(q)q} \|^{2} + \alpha \, \| \boldsymbol{q} \|^{2} = \min.$$

Then the joint velocity vector, obtained from the continuous solution of the **IKP** has the form:

(2.5) $$\boldsymbol{q} = \boldsymbol{J}^{*}\boldsymbol{x} + (\boldsymbol{I} - \boldsymbol{J}^{*}\boldsymbol{J})\,\boldsymbol{y},$$

(2.6) $$\boldsymbol{J} = \boldsymbol{V}\boldsymbol{S}^{*}\boldsymbol{U}',$$

(2.7) $$S' = \mathrm{diag}(\sigma_1/(\sigma^2{}_1+\alpha^2), \sigma_2/(\sigma^2{}_2+\alpha^2), K, \sigma_r/(\sigma^2{}_r+\alpha^2), 0, K, 0),$$

where: J^* is an **IKR** solution, obtained by means of the Singular Value Decomposition; **V, U** are left and right eigenvector matrices; **S** is a singular value's matrix; y is an arbitrary nonzero n-dimensional vector. All matrices have compatible dimensions. The second term in the right-hand side of equation. (2.5) is the projection of the arbitrary vector y onto the null space of J, since it does not result in an end effector velocity. The discretization of (2.5) give us the following sampled-data equation:

(2.8) $$q_{j+1} = J^*(q_i)\,(\mathbf{x}_{i+1} - \mathbf{x}_i) + [\,\mathbf{I} - J^*(q_i)\,J(q_i)]\,y + q_i.$$

The equation (2.8) is basic for the Kinematic Control Algorithm (**KC**). It represents the relation between the one-step-predicted joint position vector, the variation of the desired Cartesian position vector and the actual joint position vector. The proposed **KC** method does not require the specification of Cartesian velocities, so it can be applied for predictive control of redundant arm joint positions. This method provide solution with function continued for every space of manipulation and performs given optimisation criteria with a null space solution. The solution of the differential equation (2.5) contains two components: the first term is a particular solution and the second term is a homogeneous solution. The term of the homogeneous solution is frequently used to optimise some secondary criterion under the constraint of the specified end effector velocity by choosing y to be gradient of some prescribed function $H(q)$. The homogeneous solution can also be used to optimise secondary criteria defined in Cartesian space to avoid obstacles. Similar to the collision free path planning algorithm, based on a potential field $H(q)$, we can define a potential field over a task space, such that obstacles can be represented as high potentials regions. Collision-free path planning of a redundant arm can be done by searching for a path of minimal potentials under constraints of arm kinematics as well as in compromise with other performance indices. We assume that an obstacle is a hipper-sphere of radius **r** with its centre located at obstacle coordinates x. Then a potential field $H(q)$, can be defined for collision-free path planning of the end-effector under a single obstacle, as follows:

(2.9) $$H_{\mathrm{obst}}(q) = \frac{1}{1 + e^{X/T}},$$

where T is a positive constant and $X(q)$ is a quadratic function:

(2.10) $$X(q) = \| x(q) - x_{\mathrm{obst}} \|^2 - r^2.$$

In the case where many obstacles occur in the task space, and/or many arm-body-points are subject to collision avoidance, H_{obst} should be formed as a sum of all the potential fields defined for individual obstacles and/or for individual arm-bodypoints.

2.2. NN for kinematic control

The **NN** approximation of the nonlinear functions in the equation (2.8) requires the actual joint position q_i and the desired Cartesian position x_{i+1} as inputs to calculate the one step prediction of the joint configuration. This mapping is highly nonlinear, since the transformation depends on robot configuration and its kinematic structure.

The context-sensitive **NN (Fig. 5),** contains two **NN**s, named **Context** and **Function**. The **Context NN** consists of nonlinear units and a **BP** learning process, whose outputs are used to set up the weights of the **Function NN**. The first **NN** has many outputs as there are weights of the second **NN**. Since the number of outputs could be very large, the Function **NN** ought to be simple and recurrent.

The proposed solution of the **KC** problem by means of neural networks uses a context-sensitive **NN** with Cartesian feedback. That control scheme takes the advantage of the functional

decomposition of robot kinematics, as it reflected on the structure of the equation (2.8). The functional part of the context-sensitive **NN** realises the product between the desired Cartesian position and the **JPF**, which is robust of singularities. The Context **NN** is a three layered feedforward **NN** with a **BP** learning. The hidden layer of this **NN** is based on Gaussian Activation Function. The output layer adjusts the corresponding weights of the Functional **RNN**. The structure of the **KC** system and some simulation results of six-link planar robot-manipulator avoiding obstacle, are given in [7] and shown on Fig. 5.

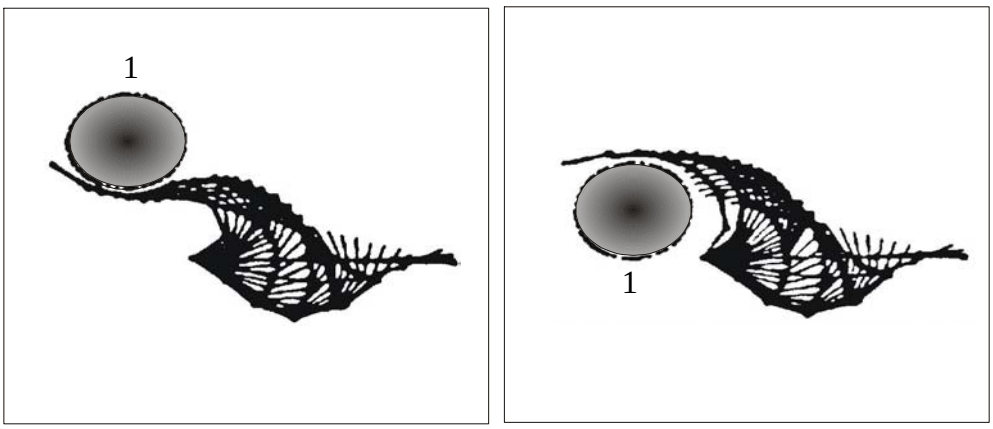

Fig. 5. Simulation of obstacle avoidance

3. Conclusion

The paper describes the last achievements of the scientists from the **PIS dept., IIT**, **BAS** in the field of **NN** theory and applications. All theoretical approaches are illustrated by experimental results and appropriate examples.

APPENDIX [2]

Table. A1.1. **NN** architectures

Feedforward Nns			**Feedback NN**s (**RNN**)			
Single layer perceptron	Multilayer peceptron	Radial Basis Function net	Competitive network	Kohonen's SOM	Hopfield network	ART models

Table A1.2. **NN** learning algorithms

Paradigm	**Learning rule**	**Architecture**	**Learning algorithm**	**Task**
Supervised	Error-correction	Single- or multilayer perceptron	Perceptron LA BP	PR, classification, AF,PP I&C,(Adaline&Madaline)
	Boltzmann	Recurrent	Boltzmann learning	PR,classification
	Hebbian	Multilayer feedforward (FF)	Linear discriminant analysis	Data analysis(DA) PR,classification
	Competitive	Competitive	Learning vector	Within-class categorization (WCC) Data compression (DC)
		RT network	ART map	PR, classification , WCC
Unsupervised	Error-correction	Multilayer FF	Sammon's projection	DA
	Hebbian	Feedforward	Principal component or competitive analysis	DA DC
		Hopfield	Associative memory	Associative memory
	Competitive	Competitive	Vector quantization	Categorization, DC
		Kohonen's	Kohonen's SOM	Categorization, DA
		ART networks	ART1, ART2, ART3	Categorization
Hybrid	Error-correction & competitive	RBF network	RBF learning algorithm	PR, classification AF, PP, I&C

Legend: **AF** – Approximation of functions**; PR** – Pattern recognition; **PP** – Process (time series) prediction; **I&C** – Identification and control.

References

1. Simon H. Neural Networks. Macmillan Publ. Company,1994.
2. Jain, A.K. Artificial neural networks: a tutorial, computer. – In: IEEE, March 1996, 31-44.
3. Zeidenberg, M. NNs Models in Artificial Intelligence. USA, Ellis Horwood, 1990.
4. Lippmann, R. P. An Introduction to Computing with NN. – In: IEEE ASSP Mag., April, 1987, 4-5.
5. Baruch, I, I. Stojanov, E. Gortcheva. Recurrent trainable neural networks: topology and learning. – In: Proc. of the 5 Int. Symp. TAINN'96, 27-28 June 1996, Istanbul, Turkey, 40-49.
6. Baruch, I, I. Stojanov, E. Gortcheva. Neural Network Models of Dynamic Processes: Stability and Learning. – In: Proc. of the 3th Int. Symp. MMAR'96, Sept.10-13,1996, Miedzyzdroje, Poland, vol.3, 1169-1174.
7. Gortcheva, E., J. M. Ibarra-Zannatha, I. Baruch, I. Stoyanov. Kinematic neural control of redundant robots, avoiding obstacles. – In: Proc. of the UNESCO Int. Conf. ICRAM'95, 14-16 Aug. 1995, Istanbul, Turkey, vol. II, 906-910.
8. Stojanov, I. An Improved Backpropagation NN Learning. – In: Proc. of the 13th Intern. Conf. on PR, TU, 25-29 Aug. 1995, Vienna, Austria. IEEE Computer Soc., vol.II, Track B, PR and SP, 586-588.

Приложение рекуррентных нейронных сетей в системах идентификации и управления

Йерохам Барух, Александр Митев, Бойка Ненкова

Институт информационных технологий, 1113 София

(Р е з ю м е)

Представлены непрерывные и дискретные математические модели рекуррентных нейронных сетей. Описана их двухслойная архитектура и ее применение при идентификации линейных и нелинейных динамических процессов.

Показаны результаты экспериментов при аппроксимации функций и при кинетическом обучении роботов. В приложении представлена классификация нейронных сетей по отношении топологии и подход обучения.