NEURAL NETWORK BASED ALGORITHM FOR DYNAMIC SYSTEM OPTIMIZATION

Roseli Francelin Romero, Janusz Kacprzyk and Fernando Gomide

ABSTRACT

A class of artificial neural networks with a two-layer feedback topology to solve nonlinear discrete dynamic optimization problems is developed. Generalized recurrent neuron models are introduced. A direct method to assign the weights of neural networks is presented. The method is based on Bellmann’s Optimality Principle and on the interchange of information which occurs during the synaptic chemical processing among neurons. A comparative analysis of the computational requirements is made. The analysis shows advantages of this approach as compared to the standard dynamic programming algorithm. The technique has been applied to several important optimization problems, such as shortest path and control optimal problems.

KeyWords: Neural networks, dynamic programming, control optimal.

I. INTRODUCTION

Bellman’s dynamic programming [5] is one of the most powerful techniques developed for the solution of optimization problems. At least in principle, this technique solves many important decision problems in fields such as electrical engineering, aerospace engineering, chemical engineering, economics, operations research and artificial intelligence.

Learning methods based on dynamic programming (DP) have received increasing attention in artificial intelligence. Researchers have argued that DP provides the appropriate basis for providing planning results that will determine reactive strategies for real-time control, as well as for learning such strategies when the system being controlled is not well-known [4].

Because of the high computational requirements of the standard DP algorithm, algorithms based on DP (and more specifically, learning algorithms) employ novel means for improving the computational efficiency of conventional DP algorithms. They can be divided in two major classes: synchronous DP algorithms for which the evaluation function at a certain stage depends on the evaluation function obtained in the previous stage, and asynchronous DP algorithms for which this does not apply.

Most learning algorithms based on DP require prior knowledge of the system underlying the decision problem. If the system is deterministic, as in the case considered in the present work, one must know the successor states and the immediate costs for all admissible actions for every state. When this knowledge is not available the problem is known as a decision problem with incomplete information; techniques for obtaining this information are called adaptive methods.

Adaptive methods can be indirect or direct methods. An indirect method consists of modeling the system being through an identification phase, and in the determination of the decision policy [4,19]. A direct method determines only the decision policy from a known model of the decision problem [8,20,21]. An important direct method, which has been extended by many researchers, is the Q-learning algorithm proposed by Watkins [26]. This learning algorithm is not deterministic and is not adaptive in the usual sense, because it does not depend on a training data consisting of input vectors and the corresponding desired output vectors. Instead, it generates actions that are not strictly viewed as responses to input patterns, but it depends also on system states.

This paper proposes an artificial neural network constituted by a two-layer feedback topology to solve non-
linear discrete dynamic optimization problems. Generalized recurrent neuron models are introduced. A direct method to assign the weights to the neural network is presented. The method is based on Bellman’s Optimality Principle and on the interchange of information which occurs during the synaptic chemical processing among neurons. The artificial neural network proposed also solves fuzzy decision making problems. The architecture developed does not require iterative training algorithms for the weights adaptation. Prior knowledge is directly and systematically stored in the neural network in an unsupervised manner.

The neural network based algorithm is an advantageous approach for DP due to the inherent parallelism of the neural networks; further it reduces the severity of computational problems that can occur in methods like conventional DP methods and in particular in Q-learning. The paper is organized as follows. First, the decision-making problem to be addressed is stated in Section 2. Next, the neural network characteristics are introduced and a computational requirements analysis is performed in Section 3. The details and use of the proposed neural network are illustrated through an application example in Section 4. Finally, the conclusions and future works are addressed in Section 5.

II. OPTIMIZATION AND DYNAMIC PROGRAMMING

In this paper, we address the problem of designing an intelligent device (a controller, a decision maker, etc…, depending on the application considered) that drives, in \( N \) temporal stages, the state vector of a determinist dynamic system from any initial point to any final point so as to minimize a certain objective function. We assume that a system model is available in the form of a nonlinear discrete-time dynamic equation

\[
x(k + 1) = f_{N-k}(x(k), u(k)), \quad k = 0, 1, \ldots, N - 1;
\]

\[
x(k) \in \mathbb{R}^n, \quad u(k) \in \mathbb{R}^m. \tag{1}
\]

It is desired to determine a decision policy that minimizes a performance index of the form,

\[
J = h(x(N)) + \sum_{k=0}^{N-1} g(x(k), u(k)) \tag{2}
\]

where \( N \) is the specified final stage and the function \( g \) is assumed to be additively separable, i.e., \( g(x(N-k), u(N-k)) = G(x(N-k)) + \tilde{G}(u(N-k)) \). The equation (1) and the performance index (2) may either be a discrete approximation of a continuous system or may represent a system that is actually discrete. It is well known [15] that, by applying DP to this problem (adopting the backward procedure), the following recurrence equation holds:

\[
J_{N-k}^{*}(x(N-k)) = \min_{u(N-k)}[g(x(N-k), u(N-k))]
\]

\[
+ J_{N-k-1}^{*}(f_{N-k}(x(N-k), u(N-k))), \quad k = 1, 2, \ldots, N
\]

with initial value \( J^{*}(x(N)) = h(x(N)) \).

The solution of this recurrence equation is an optimal decision or an optimal policy, \( u^{*}(x(N-k), N-k), \) \( k = 1, 2, \ldots, N \), obtained by considering all admissible control values at each admissible state value. We assume the decision and state spaces discretized in \( S \) values: \( \{u_1, u_2, \ldots, u_C\} \) and in \( C \) values: \( \{x_1, x_2, \ldots, x_C\} \), respectively.

Proceeding backwards from the final \( k = N \) to the initial \( k = 1 \) stage, at each particular decision stage two phases are performed: an addition followed by a minimization. Such a flow of computation “addition at \( k = N-1 \), minimization at \( k = N-1 \), addition at \( k = N-2 \), minimization at \( k = N-2, \ldots, \) addition at \( k = 1 \), minimization at \( k = 1 \)” can be modeled by the neural network presented in the next section. First, notice it cannot be a traditional neural network since some “non-traditional” operations occur: the minimization or maximization over a (finite) set. Special types of neurons are needed to implement these two operations. Fortunately, models of such neurons may be obtained as special cases of the generalized recurrent neuron introduced below.

III. AN ARTIFICIAL NEURAL NETWORK MODEL

Various models of artificial neurons have been proposed in the literature. Most generalize the basic functions of the neuron proposed by McCulloch and Pitts [17] in 1943. The necessity of developing these models occurs because sometimes it is desirable to obtain new structures of networks with nonlinearities and with different operators. Thus new learning policies need to be worked out. In this paper, a generalized model neuron, essential for building the neural network based algorithm proposed herein is introduced as follows.

3.1 A generalized neuron model

The artificial, generalized neuron Fig. 1 is assumed to be a computational device which:

- aggregates its \( n \) inputs \( a_i \)

\[
u = \sum_{i=1}^{n} g_i(a_i) \tag{4}
\]

according to the weighting functions of the synapses linking the input (pre-synaptic) neurons \( n_i \) the post-synaptic neuron \( n_k \), where the input \( a = (a_1, a_2, \ldots, a_n) \) into the neuron \( n_k \) is transformed by the weight function \( G \) as follows:
Two axonic thresholds can be defined such that if the neuron is a McCulloch and Pitts neuron [17]. If we assume (5) and \( f: V \mapsto [0, 1] \), two axonic thresholds can be defined such that

\[
y = f(\Sigma g_i(a_i))
\]

where \( g_i: R \mapsto R \) and \( a \) is an aggregation operator. Note that the weight function \( G \) is a function \( G: R^\prime \mapsto R^\prime \).

- encodes \( v \in V \) into the axonic activation \( y \) of the post-synaptic neuron:

\[
y = \begin{cases} 
  f(u) & \text{if } u \geq \alpha \\
  0 & \text{otherwise}
\end{cases}
\]

where \( \alpha \) is the axonic threshold and \( f \) is an activation function or transfer function.

The generalized neuron has a recurrent synopsis, established if the axon of neuron \( n_i \) makes contacts with dendrites or the cell body of \( n_i \) itself. If the recurrent synopsis is located near the axon, then it may control the axonic threshold as a function of the \( n_i \)'s activity [22]. Based on this definition, we note that:

If \( g_i(a_i) = c_i a_i \), where \( c_i \) is a constant

\[
a = (a_1, a_2, \ldots, a_n) \mapsto (g_1(a_1), g_2(a_2), \ldots, g_n(a_n))
\]

then the neuron is a binary neuron, that is, a McCulloch and Pitts neuron [17]. If we assume (5) and \( f: V \mapsto [0, 1] \), two axonic thresholds can be defined such that

\[
y = \begin{cases} 
  1 & \text{if } u \geq \alpha_1 \\
  f(u) & \text{if } \alpha_1 \leq u \leq \alpha_2 \\
  0 & \text{otherwise}
\end{cases}
\]

where \( \alpha_1 \) and \( \alpha_2 \), in many applications are furnished by a special type of neuron, called bias cell. Then, the neuron obtained is called a fuzzy neuron [12, 13]. Note that the generalized neuron model embodies most of the current models, including the usual sigmoidal neuron. For our purposes, however, the following models are of interest.

**Generalized Max-Neuron** can be obtained. Using (5) and taking the axonic threshold to be \( \alpha_i(\tau) = 0 \) at \( \tau = 0 \) in (6) as determined by the setting neuron, and at \( \tau \) it is set equal to the firing level \( y(\tau - 1) \) it is set equal to the firing level \( y(\tau - 1) \) we get the output \( y(t) \):

\[
y(t) = \begin{cases} 
  \alpha_i(\tau) & \text{if } u(\tau) \leq \alpha_i(\tau) \\
  u(\tau) & \text{otherwise}
\end{cases}
\]

where \( u(\tau) \) is the post-synaptic activation at \( \tau \). Then, the output \( y(\tau) \) of the neuron at \( \tau \) encodes

\[
y(\tau) = \bigvee_{i=1}^n g_i(a_i) = \bigvee_{i=1}^n c_i a_i \tag{7}
\]

where \( \bigvee \) is the maximum operator. Furthermore, the Max-Neuron, proposed by Gomide and Rocha [13] can also be obtained if \( g_i = I \) for all \( i \), then: \( y(\tau) = \bigvee_{i=1}^n a_i \) where \( I \) denotes the identity function.

**Generalized Min-Neuron** can be obtained if (5) is valid and the axonic threshold \( \alpha_i(\tau) = 1 \) at \( \tau = 0 \) in (6) (by the setting neuron), and at \( \tau \) \( \alpha_i(\tau) = y(\tau - 1) \). The output \( y(t) \) is given by

\[
y(t) = \begin{cases} 
  \alpha_i(\tau) & \text{if } u(\tau) \geq \alpha_i(\tau) \\
  u(\tau) & \text{otherwise}
\end{cases}
\]

where \( u(\tau) \) is the post-synaptic activation at \( \tau \). In this case, the output \( y(\tau) \) of the neuron at time \( \tau \) encodes

\[
y(\tau) = \bigwedge_{i=1}^n g_i(a_i) = \bigwedge_{i=1}^n c_i a_i \tag{8}
\]

where \( \bigwedge \) is the minimum operator. In the same way, the Min-Neuron, proposed by Gomide and Rocha [13] can be obtained if \( g_i = I \) for all \( i \), then: \( y(\tau) = \bigwedge_{i=1}^n a_i \) where \( I \) denotes the identity function.

The DP Neural Network (DPNN) has a feedback topology composed of two layers Fig. 2. The first layer neurons consist of generalized neurons, called U-neurons; the second layer neurons consist of generalized Max-Neurons (or Min-Neurons), called here X-neurons. The first and second layers are called layer-U and layer-X, respectively.

Inputs to the first layer neurons correspond to the different values that \( u(k) \) can assume at stage \( N - k \) whereas the inputs to the second layer neurons correspond to the different values that \( x(k) \) can assume at stage \( N - k \).
The inputs of the first layer and the second layer neurons are weighted by the weighting functions \( G \) and \( H \), respectively. U- and X-neurons are defined next.

**U-neuron** Fig. 3 is a generalized neuron that receives, at a given stage \( k \), two synchronized inputs, \( a \) and \( b \), to provide an output \( s \). These inputs can be vectors of suitable dimensions. Assuming the inputs \( a = (a_1, \ldots, a_n) \) and \( b = (b_1, \ldots, b_n) \) \( \in \mathbb{R}^n \) and the weighting functions \( G(a) = (g_1(a_1), g_2(a_2), \ldots, g_n(a_n)) \) and \( H(b) = (h_1(b_1), h_2(b_2), \ldots, h_n(b_n)) \) respectively, the output \( s \) is given by:

\[
s = f(\sum_{i=1}^{n} g_i(a_i) + \sum_{i=1}^{n} h_i(b_i)) \]

where

\[
f(\theta) = \begin{cases} 
\theta & \text{if } \theta \geq \alpha \\
0 & \text{otherwise} 
\end{cases}
\]

and \( \alpha \) is a threshold value (a similar function has been used in the ASE procedure proposed in [3]). The input \( a \) is interpreted here as one among the possible values that variable \( u \) assumes at stage \( N - k \), whereas the weighting function \( G(a) \) will correspond to that part in (3) related to variable \( u \) at stage \( N - k \). These can be provided by the network designer. The input \( b \) and the weighting function \( H(b) \) are interpreted as follows:

- for the \((N - 1)^{th}\)-stage, the input \( b \) corresponds to one among possible values of variable \( x(N) \) and \( H(b) \) corresponds to \( h(x(N)) \). This value is also known and provided to the network during the initialization phase.
- for the \((N - k)^{th}\)-stages, \( k = 2, \ldots, N \), the input \( b \) is unidimensional because it is the output of some X-neuron thus, the weighting function \( H(b) = I \), where \( I \) denotes the identity function.

**X-neuron** is either a generalized Max-Neuron or generalized Min-Neuron, depending on the optimization problem. It receives synchronized inputs: \( \nu_i \) to provide an output \( s \). To solve problem (2) the X-neuron will be considered as a Min-Neuron Fig. 3. Assuming \( a = (a_1, \ldots, a_m) \) and \( b = (b_1, \ldots, b_m) \) \( \in \mathbb{R}^m \) the inputs of a determined X-neuron and the weighting functions

\[
G(a) = (g_1(a_1), g_2(a_2), \ldots, g_m(a_m))
\]

and

\[
H(b) = (h_1(b_1), h_2(b_2), \ldots, h_m(b_m))
\]

respectively, the output \( s \) is:

\[
s = \min_{i} \nu_i(\nu_i)
\]

where \( \nu_i = G(a) + H(b) \), and \( n \) denotes the number of X-neuron inputs. For the purpose of this paper, the input \( a \) is interpreted as one among possible values that variable \( x \) assumes at stage \( N - k \), whereas the weighting function
$G(a)$ corresponds to that part in (3) related to variable $x$ at stage $N-k$. Therefore, these can be also set by the network designer. The input $b$ is unidimensional because it is the output of some U-neuron and the corresponding weighting function is $H(b) = 1$. Hence, the X-neuron, as a generalized Min-Neuron, is responsible for the minimization stated in (3).

That part of the objective function related to $J'$, that is, the optimal value obtained from the previous stage (that is the $(N-(k-1))^{th}$ stage) is determined by the feedback connections of the proposed network. The term $f(x(N-k), u(N-k))$, $k = 1, 2, \ldots, N$, of (3) is computed based on the concept of controller function (borrowed from biology) described below, as a method for weight assignment.

### 3.2 Method for weights assignment

To proceed with the development of the proposed neural network, the concepts of amounts of transmitters, receptors and controllers described in [22] are of interest; they are associated here with values of decision variables, state variables, and constraints of the decision problem.

Let $T$ denote a neuron transmitter, $R$ its receptor and $C$, its controller. It is known that the activation of the post-synaptic cell $n_i$ due to transmitter $T$ released by the pre-synaptic cell $n_i$ activates control molecules $C_i$. This neuronal chemical processing can be modelled symbolically by

$$T_i \odot R_j >> C_j$$  (10)

where $\odot$ and $>>$ denote concatenation and translation operations. Each specific coupling between a pre-synaptic transmitter and a post-synaptic receptor, in turn, activates different types of controllers $C_j$. These controllers exercise different types of action over the pre- and post-synaptic cells,

$$T_i \odot R_j >> C_j \rightarrow \text{action}$$  (11)

For addition, the amount $q(C)$ of controllers triggered by $T \odot R$ binding is calculated as:

$$q(C) = (q(T) \odot q(R)) o \mu(T, R)$$  (12)

where $q(T)$ is the amount of $T$, $q(R)$ is the amount of $R$, $\mu(T, R)$ is the degree of matching between $T$ and $R$, $\odot$ and $o$ are, in general, t-norms or s-norms [22,27], when $q(T)$, $q(R)$, $\mu(T, R) \in [0, 1]$.

To simplify the exposition, we summarize the equations (10)-(12) as

**Initialization Pass**

Let $q(T(X))$ and $q(T(U))$ be the transmitter quantities of X-and U-neurons, respectively. Similarly, let $q(R(X))$ and $q(R(U))$ be the receptor quantities of X-and U-neurons. Assign the discrete values of variables $a$ and $x$ in each stage $N-k$ with these quantities as follows $q(R(U)) = u_i(N-k)$ and $q(R(X)) = x_i(N-k)$ for all $(i^{th})$ U-neuron and $(i^{th})$ X-neuron in U-layer and X-layer, respectively. For simplicity, suppose $q(T(X)) = q(T(X'))$ for all $(i^{th})$ X-neuron and $\mu(T, R) = 1$. Let $q(C(U))$ be the controller quantity of the U-neuron and $W(X, U)$ be the weighting value between the X-neuron and U-neuron.

**Symbolic Processing Pass**

During this pass the neurons of the neural network exchange messages before establishing connections and corresponding weights. An X-neuron sends a message $q(T(X))$ to a U-neuron (Fig. 4). The U-neuron decodes this message through its receptor $q(R(U))$ and releases its controller, $C(U)$, that is, it computes:

$$q(C(U)) = f(q(T(X)), q(R(U))$$

where $f$ is as in (1). Hence, the controller amount $q(C(U))$ corresponds to the calculation of $f(x(N-k), u(N-k))$ in (3). Based on value $q(C(U))$ the following test is performed:

**If** ($q(C(U))$ is feasible) **then** a connection from U-neuron to X-neuron is established and the corresponding weight is set equal to $W(U, X) = 1$

**else** $W(U, X) = 0$.

During this process the following actions can also be performed: either the U-neuron recurrent synopsis is activated and signal $b = q(C(U))$ is sent to U-neuron itself at stage $(N-1)$, or the feedback interconnections are activated and signal $q(T(U)) = q(C(U))$ is sent to all X-neurons in layer-X at stage $(N-k)$, $k = 1, \ldots, N-1$. Figure 4 illustrates the Symbolic Processing between an X- and U-neuron (the arrow line represents the message exchange).

For stage $N-k$, all U-neurons recurrent synopsis are activated; signal $b = q(C(U))$ is also sent to the U-neuron itself. The U-neuron inputs, $a = q(T(U))$ and $b = q(C(U))$ are weighted by functions $G(a)$ and $H(b)$, respectively.

For stage $N-k$, $k = 2, \ldots, N$, the network feedback

![Fig. 4. Symbolic processing between X-neuron and U-neuron.](image-url)
connection is activated and the degree of matching between a U-neuron (Fig. 5) and each X-neuron is computed, that is, the X-neuron (whose receptor amount, \( q(R(X)) \), is approximately equal to the transmitter amount of U-neuron, \( q(T(U)) \), according to some norm) is identified and its output, \( s_U \), is sent to the U-neuron. The U-neuron inputs, \( a = q(R(U)) \) and \( b = s_X \) are weighted by functions are \( G(a) \) and \( H(b) = I \), respectively.

**Numeric Processing Pass.** The numeric calculation, henceforth called Numeric Processing, is performed after the Symbolic Processing is completed and is as follows:

1 - calculation of U-neuron output, \( s_U \), at each stage, that is,

\[
s_U = f(G(a) + H(b)) = f(\sum_i g_i(a_i) + \sum_i h_i(b_i))
\]

where the \( H(b) = I \) for stage \((N - k), k = 1, \ldots, N - 1 \).

2 - calculation of X-neuron output at each stage, that is,

\[
s_X = \max_i \{ \nu_i \}
\]

where \( \nu_i \) is computed as a function of the output of some U-neuron, \( b = s_U \), and its other input \( a = q(T(X)) \) and is given by \( \nu_i = I(b) + \sum g_i(a_i) \). Figure 6 illustrates the Numeric Processing among the first X-neuron and all U-neurons.

### 3.3 Processing algorithm

The network processing may be portrayed in the following general form:
Main Algorithm

Begin
k = 1
determine \( \hat{h}_i(b) \) for each U-neuron, \( i = 1, \ldots, N \)

While (\( k \leq N \)) do

Perform Initialization Pass
Perform Symbolic Processing (\( N - k \))-Pass for pair U-neuron and X-neuron

If \( W(U_i, X_j) = 1 \) then

Perform Numeric Processing Pass
Save the value of the variable \( u \) corresponding to output from each X-neuron \( k = k + 1 \)

End

Next, a theorem establishing the correspondence between the DP algorithm and the weight assignment and processing schemes introduced in Sections 3.2 and 3.3 is presented.

3.4 Equivalence theorem

Theorem. The solution provided by Neural Network and Processing Algorithm as defined in Section 3.3 is equivalent to the Dynamic Programming Algorithm.

Proof. Without the loss of generality, assume that a DP problem is solved by backward recursion. To show its equivalence, we will show \( J_{N-1,N} \), \( k = 1, 2, \ldots, N \) given by (3) are also processed by the DPNN.

Recall that to construct the DPNN we know that for each variable decision \( u_{k-1} \) and state variable \( x_{k-1} \) there is a corresponding U-neuron and a X-neuron, respectively, at the stage \( (N - k) \). The proof is made by induction.

a) First, we show that the activation of U-layer and X-layer processing corresponds to the calculations done at stage \( N - 1 \) in (3), when \( k = 1 \), that is:

\[
\min \{ h(x(N)) + g(x(N-1), u(N-1)) \}
\]

is computed. Assuming that \( g(x(N-1), u(N-1)) \) additively separable, i.e., \( g(x(N-1), u(N-1)) = \tilde{G}(u(N-1)) + \tilde{G}(x(N-1)) \), we have

\[
\min \{ h(x(N)) + \tilde{G}(x(N-1)), \tilde{G}(u(N-1)) \} \quad (13)
\]

which must be calculated when \( k = 1 \). According to the algorithm, functions \( \hat{h}_i(b) \) must be assigned for all \( i^{th} \) U-neuron. These functions, by definition, correspond to function \( h(.) \) in expression (13). Next, the Initialization Pass is performed. In this pass, the inputs related to stage \( N - 1 \)

\[
a = q(R(U_i)) = u_i(N-1) \quad \text{for} \quad 1 \leq j \leq S;
\]

\[
a = q(T(X_j)) = x_j(N-1) \quad \text{for} \quad 1 \leq j \leq C;
\]

and the corresponding weighting functions are also assigned. Values \( u_i \) and \( x_j \) are the values that the variables \( u \) and \( x \) assume at stage \( N - 1 \), respectively. The weighting functions are taken equal to \( \tilde{G}(u(N-1)) \) and \( \tilde{G}(x(N-1)) \), by definition.

The Symbolic Processing \( N - 1 \) Pass is performed next. In this Pass, the connections between U-layer and X-layer must be established. The controller function

\[
q(C(U_i)) = f_0 \cdot (q(T(U_i)), q(R(X_i)))
\]

is calculated, where \( f_0 \) is given in (1). This corresponds to calculating \( f_{k-1}(u(N-1)), u(N-1) = x(N) \). If \( q(C(U_i)) \) is feasible then \( W(U_i, X_j) = 1 \) and since \( k = 1 \) (stage \( N - 1 \)), the recurrent connection is activated by \( b = q(C(U_i)) \), which weighting function is \( \hat{H} \). (Recalling: Fig. 5 shows the Symbolic Processing among the first X-neuron and all U-neurons. In this figure, there is no connection between the first X-neuron and first U-neuron, represented by the arrow line). As this processing is held for all \( i^{th} \) U-neuron, the value \( h(x(N)) \) in the expression (13) is obtained. Following the Main Algorithm, if \( W(U_i, X_j) = 1 \) (see in Fig. 5, \( W(U_i, X_j) = 1 \)), the Processing Numeric Pass must be executed. In this pass, the output of each \( i^{th} \) U-neuron, \( s_U \) is

\[
s_U = f(\tilde{G} + \tilde{H}) = f(\sum g_i(a_i) + \sum \hat{h}_i(b_i))
\]

which corresponds to the value \( \tilde{G}(u(N-1)) + h(x(N)) \) in the expression (13).

Next, the \( (i^{th}) \) X-neuron, in X-layer, receives \( b = s_U \), weighted by the function \( H = 1 \) and its remaining input \( a = q(T(X_i)) \), weighted by the function \( G \), and computes:

\[
v_i = \sum g_i(a_i) + H(b) = \sum g_i(a_i)b
\]

for each \( (i^{th}) \) U-neuron. This corresponds to the term \( h(x(N)) + G(x(N-1)) \) in the expression (13). When all of the inputs in the \( (i^{th}) \) X-neuron are calculated, its output is

\[
s_x = \min_i v_i.
\]

Thus (13) is computed. Figure 6 shows the Numeric Processing among the neurons \( X_i \) and all U-Neurons supposing \( W(U_i, X_j) = 0 \).

b) Now, supposing that for \( k = p - 1 \) the neural network performs the desired computations at stage \( N - p - 1 \), it should be shown that it also performs the correct computations at stage \( N - p \) when \( k = p \). That is, it should be shown that

\[
J_{N-p,N} = \min_{u(N-p)} \{ g(x(N-p), u(N-p))
\]

\[
+ J_{N-(p-1),x(N-(p-1))} \}
\]

(14)
is computed by the DPNN. From the induction hypothesis, assume that $J_{N-\phi-1,N}(x(N-(p-1)))$ has been obtained. Hence, it remains to be shown that the first parcel of (14) is calculated when $k=p$ and that the minimization is also performed. In fact, according to the Main Algorithm, in the Initialization Pass, the following values related to stage $N-p$ are set:

$$q(T(U_j)) = u_i(N-p) \text{ for } 1 \leq j \leq S;$$

$$q(T(X_j)) = x_j(N-p) \text{ for } 1 \leq j \leq C;$$

and the corresponding weighting functions are also assigned. Recall that $u_i$ and $x_j$ are values that the variables $u$ and $x$ assume at stage $N-p$, respectively. The weighting functions are taken as $G(u(N-p))$ and $G(x(N-p))$, by definition.

The Symbolic Processing $N-p$ Pass is performed next. The connections between U-layer and X-layer must be established. The controller function

$$q(C(U_j)) = f_{N-(N-p)}(q(T(U_j)), q(R(X_j)))$$

is calculated, where $f_{N-(N-p)}$ is given in (1). This corresponds to finding $f_{N-p}(x(N-p), u(N-p)) = x(N-(p-1))$.

If $q(C(U_j))$ is feasible then $W(U_j, X_j) = 1$ and since $k \neq 1$, the network feedback connection is activated and message $q(C(U_j))$ is sent to all X-neuron of X-layer. The X-neuron with the highest degree of matching into the $i^{th}$U-neuron, is determined, i.e., the X-neuron whose transmitter quantity is equal to $q(C(U_j))$ is activated and its output, $s_x$, is received by the $(i^{th})$U-neuron, $b = s_x$. This can be seen in Figure 5, where it is supposed that the $(i^{th})$X-neuron has been the winner neuron, that is, $q(T(X_j)) = q(T(U_j))$. In this case, the input $b$ of the U-neuron, $U_2$, is set equal to $b = s_x$. This occurs for all $(i^{th})$U-neuron. In this way, the value $J_{N-\phi-1,N}(x(N-(p-1)))$ obtained at previous stage is considered in this stage because $s_x = \max\{u_i\}$ (this corresponds to $J$ calculated at a previous stage).

Following the Main Algorithm, if $W(U_j, X_j) = 1$, the Processing Numeric Pass is performed and the output of each $(i^{th})$U-neuron, $s(U_j)$ is found, for all $i$, by

$$s_{U_j} = \sum_i \tilde{g}_i(a_i) + b$$

which corresponds to compute $G(u(N-p)) + J_{N-\phi-1,N}(x(N-(p-1)))$.

Next, the $(j^{th})$X-neuron receives this signal, $b = s_{U_j}$ and its other input $a = x_j$. The corresponding weighting functions are the $H = I$ and $G$, respectively. Thus, the output to the $(j^{th})$X-neuron is

$$v_j = \sum_i \tilde{g}_i(a_i) + I(b)$$

for all $(i^{th})$U-neurons. Therefore the value $G(x(N-p)) +$ $\tilde{G}(u(N-p)) + J_{N-\phi-1,N}(x(N-(p-1)))$ is calculated. This is analogous to determine $g(x(N-p), u(N-p)) + J_{N-\phi-1,N}(x(N-(p-1)))$ value. When all of the inputs to $(j^{th})$X-neuron are available, its output is given by

$$s_{X_j} = \min_i \{b_i\}.$$

Since this is valid for all $(j^{th})$X-neurons in X-layer, we readily conclude which the desired computations for stage $N-p$, and therefore, the results given by (14) are performed by DPNN and the theorem is proved.

### 3.5 Recovering the solution

According to algorithm proposed, the value of variable $u = q(R(U_j))$, corresponding to the minimum (or maximum) input in X-neuron, must be stored into X-neuron at each Stage. To follow it will be described how this can be made as well as how the optimum solution can be recovered.

It is known that the biological neuron has a memory by its transmitters and receptors. Since the different values of decision variable $u$ and state variables $x$ are associated with quantities of receptors and transmitters of biological neuron we are supposing that as U-neuron well as X-neuron have this memory.

By using the neuron mathematic model proposed by Bezdek [BEZD 91], we suggest that X-neuron model has a local memory to store the value of variable $u$ corresponding to its maximum input at each stage. Furthermore, the indice $t$ must also be stored. This is shown in Fig. 7.

According to processing of Max-neuron or Min-neuron (or X-neuron), it receives inputs synchronized in time and hence the neuron can count how much inputs it has received and to indicate the indice $t$.

![Fig. 7. Internal processing of X-neuron.](image-url)
This same mechanism is considered to U-neuron by which it stores all different values (variable u) that it receives (by its receptor) in each initialization Stage. Thanks to this synchronized mechanism, it can take these values successively as at the determination as at recovery of the solution. In forward sense, this mechanism takes the values from left to right and in backward sense, it takes the values from right to left.

Figure 7, the output of X-neuron, in sense forward, denoted by \( s_f = s(X) \) is given by:

\[
s_f = a_i(X) = \min_i(a_i(X))
\]

being \( a_i(X) = s(U^i) \) for some \( k^\text{th} \) U-neuron in Layer-U. Hence, if \( a_i(X) = s(U^i) \) then the value \( u' = q(T(U^i)) \) must be stored as the optimal of variable u corresponding to state variable x. On the other hand, the output of X-neuron, in sense backward, denoted by \( s_b = s(X) \) is given by:

\[
s_b = \{q(T(U^i)), t\} = \{u', t\}
\]

The solution can be recovered in this following way. When all stages are performed we have the optimal value of the objective function. From this value, the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal value of the objective function is furnished by an X-neuron and this has a local memory storing the corresponding value of the objective function. From this value, the optimal solution, that is the set of variables u at each stage, can be recovered in this following way. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward. First, as the optimal solution, that is the set of variables u at each stage, can be recovered in the sense backward.

3.6 Discussion about learning and the algorithm proposed

Four interesting facts can be stated about scheme proposed here. The first concerns the establishment of connections from U-neurons to X-neurons. During this phase, the Symbolic Processing computes values \( C(U) \) for all state-action pairs and makes use of these values to select actions. An action here can be either the activation of the controller function or the activation of connections. This scheme is similar to the one that occurs in a Q-learning system [26], in which estimations of \( \overline{Q} \) are obtained for all state-action pairs and the actions are selected based on these estimations. The second interesting fact concerns the way input \( b \) is chosen when the network feedback connection is activated. This input is the closest to the amount \( q(T(U)) \) according to the euclidian norm. Therefore, DPNN performs a procedure similar to the one used by Kohonen [16], known as competitive learning. Third, the weights of the network can be assigned in parallel, that is, when a X-neuron is connected with all U-neurons, the remaining X-neurons can also simultaneously do the same. The X-neuron outputs can also be computed in parallel, providing the learning algorithm characteristics of parallelism not available in the traditional DP algorithm and its variation.

Finally, an analysis of computational requirements should be presented to show the advantages of this approach compared to traditional DP algorithms. For this purpose, consider an optimization problem with \( N \) stages, \( \kappa \) and \( j \) discrete values for the state and decision variables respectively at stage \( i \), \( i = 1, 2, \ldots, N \).

According to [5], the DP solution requires an addition for each combination of \((u_i, x_i)\) at stages 2, \ldots, \( N \) or \( \sum_{j^{}=2}^{N} N\kappa j^{} \) additions. For each value of the state variable at all stages there are \( j_i - 1 \) comparisons, and at the last stage there are \( \kappa - 1 \) comparisons to calculate the minimization. This produces total of \( \sum_{j=2}^{N} \kappa(j_i - 1) + \kappa - 1 \) comparisons.

Assume that the times spent for an addition and a comparison are \( t_A \) and \( t_C \), respectively. Then the total time \( t_{DP} \) needed for the dynamic programming solution is

\[
t_{DP} = (\sum_{j=2}^{N} \kappa j_i) N t_A + (\sum_{j=2}^{N} \kappa(j_i - 1) + \kappa - 1) t_C.
\]

For DPNN, since X-neuron computations are performed in parallel, the corresponding comparisons are also processed in parallel and hence the total time \( t_{NN} \) needed for the DPNN solution is

\[
t_{NN} = (\sum_{j=2}^{N} \kappa j_i) N t_A + N t_C.
\]

Clearly

\[
t_{NN} < t_{DP}.
\]

The DP solution total storage requirements for optimal decision and return functions, with reference to the backward recursion according to [5], is explained to follow. First, recall that \( u_{N-1}(x_{N-1}) \), \( k = 1, 2, \ldots, N \) is saved until the optimal is traced, but \( J_{N-1}(x_{N-1}) \) is saved only until \( J_{N-k} \) has been computed. Thus storage space must be available for all \( N \) functions \( u_{N-1}(x_{N-1}) \), but for only two consecutive functions \( J' \). Supposing that a decision variable assumes \( t \) values per stage, there are \( t + 2 \) tabulations of optimal functions for each value of \( x_N \), \( k \) consisting of \( s \) tabulations of \( u_{N-1} \), a single tabulation of \( J_{N-1} \) and the tabulation obtained in previous stage \( J_{N-k-1} \). Furthermore, assuming that there are \( N \) stages and \( p \) state variables per stage, each one having \( k \) feasible values, then the total storage requirements for the DP solution is

\[
t_{DP} = N(t + 2) k^p.
\]

In the neural net model case, since all information is saved in net topology, a network constituted by two layers with feedback with \( k^p \) X-neurons and \( t \) U-neurons is needed to obtain the optimal solution. Recall that in a
neural network what is stored is the interconnection matrix. Hence storage space must be available for $S_{NN} = n^t \times k^p$ corresponding to $t$ neurons in Layer-U and $k^p$ neurons in Layer-X. Therefore,

$$s_{NN} < s_{DP}$$

This analysis shows that the proposed approach naturally embodies a parallel DP algorithm and has fewer computational requirements. In addition, since it is based on biologically plausible computations, it does provide a direct mechanism for programming neural hardware compatible with biological computation [24].

In the next section, we present an interesting application of the proposed approach. Francelin and Gomide [10] also provide an interesting example of using the algorithm presented in long-range planning of interconnections between two power systems.

IV. APPLICATIONS

In this section, we present an example to illustrate clearly how the DPNN performs.

We consider the discrete system according to [15] described by the state equation:

$$x(k+1) = A(k)x(k) + B(k)u(k)$$

where $u(k)$ and $x(k)$ can be variable satisfying constraints. The problem is to find an optimal policy $u^*(x(k), k)$ that minimizes the objective function

$$J = \frac{1}{2}x^T(N)Hx(N)$$

$$+ \frac{1}{2} \sum_{k=0}^{N-1} [x^T(k)Q(k)x(k) + u^T(k)R(k)u(k)]$$

where $H(k)$ and $Q(k)$ are real symmetric positive semi-definite $nxn$ matrices, $R(k)$ is a real symmetric positive $mxm$ matrix and $N$ is a fixed integer greater than 0.

To simplify the notation in the derivation that follows, let us assume that $A$, $B$, $R$ and $Q$ are constant matrices.

Initially we remember that the allowable state and control values must be quantized, where in the most general case, $x \in R^n$ and $u \in R^m$ for stages $k = 1, 2, ..., N$. Thus if $u = (u_1, u_2, ..., u_p)$ where $u_1$ can assume $s_1$ values, $u_2$ can assume $s_2$ values and so forth, $u_p$ can assume $s_p$ values at stage $k$ then we have that the DPNN must have $S = s_1 \cdot s_2 \cdot \cdots \cdot s_p$ neuron in U-layer, where each U-neuron will receive an $n$-dimensional input vector, denoted by $a$, which corresponds to one of the $S$ combinations of possible values for variable $a$.

Analogously, if $x = (x_1, x_2, ..., x_n)$ where $x_1$ can assume $c_1$ values, $x_2$ can assume $c_2$ values and so forth, $x_n$ can assume $c_n$ values at stage $k$ then we have that the DPNN must have $C = c_1 \cdot c_2 \cdot \cdots \cdot c_m$ neuron in X-layer, where each X-neuron will receive an $m$-dimensional input vector, denoted by $x$, which corresponds to one of the $C$ combinations of possible values for variable $x$.

We will assume, without losing generality, that the state and control, $u(k), x(k)$, respectively are quantized at the same values for all stages.

The weighting functions, $G(a), G(x)$ must be determined at each stage for each U-neuron and X-neuron in the network. According to the definition of these functions, they correspond to the terms of the objective function that depends on variables $u$ and $x$ at a given stage. Weighting function $H(b)$ corresponds to the terms of the objective function that depends on variable $x$ at stage $N$. Hence, we have:

- For $k = N – 1$

$$G(a) = \frac{1}{2}u^T(N – 1)R u(N – 1); \quad H(b) = \frac{1}{2}x^T(N)H x(N);$$

$$G(x) = \frac{1}{2}x^T(N – 1)Q x(N – 1); \quad H(b(x)) = I.$$  

- For $k = N – 2, N – 3, ..., 0$

$$G(a) = \frac{1}{2}u^T(k)R u(k); \quad H(b) = I;$$

$$G(x) = \frac{1}{2}x^T(k)Q x(k); \quad H(b(x)) = I.$$  

The processing of the network is performed according to the Main Algorithm proposed in section 3.3.

To illustrate the performance of the DPNN a constrained linear quadratic control problem (LQ) was solved. This problem has been considered by Parisini [20] where an approximate optimal solution is sought by constraining the control strategies to take on the structure of two chains (the feedback chain and the feed-forward chain) of multilayer feed-forward neural networks.

The dynamic system is:

$$x(k+1) = \begin{bmatrix} 0.65 & -0.19 \\ 0 & 0.83 \end{bmatrix} x(k) + \begin{bmatrix} 7 \\ 7 \end{bmatrix} u(k)$$

where $x(k) \in R^2$ and $u(k) \in R$. The objective function is:

$$J = \sum_{k=0}^{N-1} \mu(k) + \nu \|x(N)\|^2$$

with $\mu = 40$ and $N = 10$.

We assume that state variable $x = (x_1, x_2) \in R^2$ must satisfy the following constraints:
\[-0.5 \leq x_1(k) \leq 3.5 \quad -1.0 \leq x_2(k) \leq 1.0\]

whereas control variable \( u \in R \) must satisfy:

\[-0.5 \leq u(k) \leq 0.5\]

for all stage \( k = 0, 1, \ldots, 9 \). The variable \( x_1(k), x_2(k) \) has been discretized in 107 and 300 values respectively, then DPNN corresponding to this problem has a topology constituted by \( S = 107 \) neurons in U-layer and \( C = 300 \) neuron in X-layer. The algorithm proposed in section 3.3 was implemented in C language. The optimal neural trajectories obtained by considering two different initial points: \( x(0) = (2.5, 1.0)^T \) and \( x(0) = (2.5, -1.0)^T \) are showed in Fig. 8. The results obtained by only 10 iterations agree with those presented by [20].

V. CONCLUSION

A neural network with a two-layer feedback topology and a new learning paradigm has been presented. Generalized recurrent neuron models and an adaptive direct method to assign the weights of the network have also been proposed. The method is based on Bellmann’s Optimality Principle and on the interchange of information which occurs among neurons during the synaptic chemical processing. It has been shown that the approach proposed solves nonlinear discrete DP problems and finds an optimal solution. This approach can be adapted for solving a class of optimization problems, which solution can be obtained via DP, such as optimal control problems and fuzzy decision making problems. This approach is an advantageous approach in that it embodies a parallel algorithm for DP. To verify the effectiveness of the proposed neural network, some application examples have been presented and a comparative analysis of the computational requirements has highlighted its performance.

In the future we aim to investigate how this approach can be incorporated into a reinforcement learning algorithm (also based on the DP technique) and to investigate how it could be combined with real time indirect methods.

REFERENCES


**Roseli Aparecida Francelin Romero** received her Ph.D. degree in electrical engineering from the University of Campinas, Brazil, in 1993. From 1983 to 1988 she was a lecturer at the university Julio de Mesquita Filho (UNESP). Since abril of 1988 she is an Assistant Professor in the Department of Computer Science at the University of Sao Paulo. From 1996 to 1998, she was a Visiting Scientist Carnegie Mellon’s Robot Learning Lab. Her research interests include artificial neural networks, machine learning techniques and fuzzy logic. Dr. Romero is a member of the IEEE and of the Computer Brazilian Society (SBC).

**Janusz Kacprzyk** received his M.S. degree in computer science and automatic control in 1970 from Warsaw University of Technology in Poland. He received the Ph.D. degree in systems analysis in 1977, and the D.Sc. (“habilitation”) degrees in 1994, both from the Systems Research Institute, Polish Academy of Sciences in Warsaw, Poland.

From 1970 to present he has been employed at the Systems Research Institute, Polish Academy of Sciences, currently as full professor and Deputy Director for Research. In 1981-83, in the spring of 1986 and 1988 he was a visiting professor at various American universities.

His research interests include the use of soft computing, mainly fuzzy logic, in various areas related to intelligent systems, notably, decision making and optimization, control, database querying and information retrieval. Recently, he is working on the use of the computing with words paradigms in various areas related to the above. Professor Kacprzyk is the author of 4 books, coeditor of more than 25 volumes, and author and/or coauthor of more than 150 papers.

In 1991-1995 he was a Vice-President of IFSA, and in 1995-1999 he was a member of IFSA Board. From 1999 on he is a member of EUSFLAT Board.

Janusz Kacprzyk is the editor in chief of two book series published by the Springer-Verlag group (Physica-Verlag, Heidelberg and New York): “Studies in Fuzziness and Soft Computing” and “Advances in Soft Computing” and serves on the editorial board on more than 10 respected journals.

**Fernando Gomide** received his Ph. D. degree from Case Western Reserve University, Cleveland, Ohio, USA. He is professor of the Department of Computer Engineering and Industrial Automation of the Faculty of Electrical and Computer Engineering, State University of Campinas, Campinas, Sao Pauo, Brazil since 1983. His interest areas include fuzzy set theory and applications, computational intelligence, modeling, control and optimization, logistics and decision making systems.