Efficient On-Line Training of Multilayer Neural Controllers

Julio TANOMARU* and Sigeru OMATU*

This paper addresses the question of how to perform on-line training of multilayer neural controllers in an efficient way. At first, a plant emulator and a feedforward controller based on multilayer neural networks are described. Only a little qualitative knowledge about the plant is required. The controller must learn the inverse dynamics of the plant from randomly chosen initial weights. Basic control configurations are briefly presented. New on-line training methods, based on efficient use of memory-stored data and distinction between sampling and learning frequencies are proposed. One method, called direct inverse control error approach, is effective for small adjustments of the neural controller when it is already reasonably trained; another one, dubbed predicted output error approach, directly minimizes the control error and greatly improves convergence of the controller. Simulation results show the effectiveness of the proposed neuromorphic control structures and training methods.

Key Words: neural control, neural networks, on-line training, inverse control

1. Introduction

In recent years many efforts have been made in order to apply Artificial Neural Networks (ANNs) to control of processes. Neural or neuromorphic controllers (NCs), i.e., controllers based on an ANN structure, have been proposed to learn the inverse dynamics of the control plant from the observation of the plant’s input-output relationship through time.

The training of the ANNs involved in a control system can be performed on- or off-line, depending on whether they execute useful work or not while learning. Although off-line training is usually straightforward, conditions for assuring good generalization of the ANNs through the control space are difficult to attain, making on-line training always necessary in control applications. In fact, ideally the training should be done exclusively on-line, with the ANNs learning at high speed from any initial weights. However, normally NCs possess slow convergence, which implies poor control performance and robustness, especially during the first stages of training. There is an undeniable necessity for efficient on-line training algorithms for NCs.

This paper firstly presents the general structure of a NC based on a multilayer ANN, in which only a little qualitative knowledge about the plant is necessary. Basic control configurations are briefly presented, and new on-line training methods, based on efficient use of the available data and distinction between sampling and learning frequencies are proposed. The basic idea is to perform multiple learning operations during each single sampling period, making efficient use of available processing time and past input-output data of the control plant. Simulation results attest the effectiveness of the proposed neuromorphic control structures and training methods.

2. Neuromorphic Control Structures

Consider the discrete-time single-input-single-output process

\[ y(k+1) = f[y(k), \ldots, y(k-P+1), u(k), \ldots, u(k-Q)] \]  

where \( y \) denotes the output, \( u \) is the input, \( k \) is the discrete-time index, \( P \) and \( Q \) are non-negative integers, and \( f(.) \) is a function. In many practical cases, the plant input is bounded in amplitude, i.e., there exists \( u_M \) and \( u_m \) such that, for any \( k \),

\[ u_m \leq u(k) \leq u_M \]  

Assume that the control task is to follow some...
specified reference $r(k)$ in order to minimize some specified norm of the error $e(k) = r(k) - y(k)$ through time. Additionally, assume that $p$ and $q$, which are respectively good estimates for $P$ and $Q$, are known. Recalling that multilayer ANNs (MNNs) have basically a feedforward structure able to learn input-output mappings\(^1\), two general neural control structures are proposed.

### 2.1 Neural Plant Emulator (PE)

A MNN with $m = p + q + 1$ inputs and single output can be used for emulating $f(\cdot)$ in Eq. (1). Denoting the input-output mapping of the plant emulator by $\varphi_E(\cdot)$, and its output by $y_1$, we have

$$y_1 = \varphi_E(x_E) \quad (3)$$

where $x_E$ is an $m$-dimensional input vector to the PE. At instant $k$, the input vector is defined as $x_E(k) = [y(k), \ldots, y(k - p + 1), u(k), \ldots, u(k - q)]^T$, as shown in Fig. 1, where $z^{-1}$ stands for the unit delay operator, and $p$ and $q$ were employed as estimates of $P$ and $Q$ in Eq. (1). The PE is trained in order to minimize a norm of the error between the true output of the plant and the emulator's output, i.e., $y(k+1) - y_1$.

### 2.2 Feedforward Neural Controller (NC)

Assume that the plant in Eq. (1) is invertible, i.e., there exists a function $g(\cdot)$ such that

$$u(k) = g[y(k+1), \ldots, y(k - P + 1), u(k-1), \ldots, u(k - Q)] \quad (4)$$

Consider again a MNN with $m$-dimensional input $x_C$, output $u_1$, and input-output relationship briefly represented by

$$u_1 = \varphi_C(x_C) \quad (5)$$

where $\varphi_c(\cdot)$ denotes the mapping performed by the MNN. If, for inputs properly defined, the output of $\varphi_C(\cdot)$ approximates the output of $g(\cdot)$ in Eq. (4), the MNN can be thought of as a controller in the plant's feedforward path, as illustrated in Fig. 2. At instant $k$, the control input to the plant can be approximately obtained from Eq. (5) by setting

$$x_C(k) = [r(k+1), y(k), \ldots, y(k - p + 1), u(k-1), \ldots, u(k - q)]^T \quad (6)$$

where the reference $r(k+1)$ was used instead of the unknown $y(k+1)$. After enough training of the NC, if the output error $e(k)$ is kept small, it is possible to have

$$x_C(k) = [r(k+1), r(k), \ldots, r(k - p + 1), u(k-1), \ldots, u(k - q)]^T \quad (7)$$

emphasizing the feedforward nature of the NC. The two possible input vectors to the NC (as given in Eqs. (6) and (7)) are briefly indicated in Fig. 2.

### 3. Training Configurations

The training signal in Fig. 2 must contain the information necessary for the NC to learn the inverse dynamics of the plant. Furthermore, denoting by $J$ the error function to be minimized, it is necessary to compute the derivatives of $J$ with respect to the output of the NC, i.e., $\delta = -\partial J/\partial u_1$. The knowledge of $\delta$ suffices for updating the weights of the NC via backpropagation (BP)\(^n\). Three controller's training schemes are here briefly described\(^n\).

#### 3.1 Direct Inverse Control

This configuration, depicted in Fig. 3, can be employed for on- or off-line training. As indicated in Fig. 3, the minimization of the error signal used for training leads to the direct synthesis of the inverse plant. This is different from usual error functions in control.
systems, which are commonly based on the plant output error \( e(k) = r(k) - y(k) \). If a properly chosen NC is well trained, both the training error signal and the output error become small. However, when the NC is not well trained, small training errors do not necessarily imply small output errors.

For example, assume that \( p = P = 1 \), \( q = Q = 0 \), and that the plant input is directly connected to the output of the NC. The control configuration is the one illustrated in Fig. 2, whereas training is performed according to the scheme in Fig. 3. Consider the possible situation in which the weights of NC are such that its output is constant, i.e., independent of the NC's input vector. In such a case, it is straightforward to realize that the training error will be permanently zero (training ceases), independently of the plant output error.

In short, it can be said that the control performance of the direct inverse control configuration depends strongly on the NC's generalization ability. When the NC is trained in such a way that it is able to generalize well through the control space, good control performance can be expected. However, in general, good generalization cannot be assumed and, therefore, the configuration in Fig. 3 should not be used as the only training scheme.

At time \( k+1 \), for \( x'^{(k)} = [y(k+1), \ldots, y(k+1-q), u(k-1), \ldots, u(k-q)]^T \), the NC can be updated in order to minimize an error measure \( J \) defined as a function of the difference \( u(k) - u_i(k) \), where \( u_i(k) = \varphi_i(x'^{(k)}) \) and, therefore, the term \( \delta = -\partial J/\partial u_i \) can be easily calculated, enabling BP. For example, defining \( f(k) = 0.5[u(k) - u_i(k)]^2 \) (8) the expression for \( \delta \) (with sub-index \( k \)) becomes

\[
\delta_k = u(k) - u_i(k) \quad (9)
\]

Therefore, in the direct inverse control configuration training data for supervised learning is immediately available.

### 3.2 Direct Adaptive Control

This configuration is shown in Fig. 4. Learning is essentially on-line performed and the error \( J \) is defined upon the output error. The problem is that the exact calculation of \( \delta \) requires knowledge of the Jacobian of the plant. For example, for the error function

\[
J(k) = 0.5[e(k)]^2 \quad (10)
\]

the \( \delta \)-term is given by

\[
\delta_k = \xi_k e(k) \frac{d u(k)}{d u(k)} \quad (11)
\]

where \( e(k) = r(k) - y(k) \) and the binary factors \( \xi_k \) were introduced to account for the constraints on the input \( u(k) \). Defining \( \hat{e}(k) = e(k)[d y(k)/d u(k)] \), the factors \( \xi_k \) are given by

\[
\xi_k = \begin{cases} 0, & \text{if } \hat{e}(k) > 0 \text{ and } u(k-1) = u_M \\ 1, & \text{otherwise.} \end{cases} \quad (12)
\]

The role of \( \xi_k \) is to avoid mistraining of the NC for references that cannot be tracked. In other words, when the output error results from the physical limitation expressed in Eq. (2), \( \xi_k \) becomes zero, inhibiting learning. On the other hand, when the plant output error can be physically reduced, \( \xi_k \) becomes 1 and learning is performed in the usual way. Due to the constraints to which the input of the plant was assumed to be subjected, if \( \xi_k \) is not considered, for some references training would never stop, even if the weights of the NC are such that the output error cannot be made smaller. The effect of including \( \xi_k \) in Eq. (11) is equivalent to considering the existence of a limiter between the output of the NC and the input of the plant.

### 3.3 Indirect Adaptive Control

Although in many practical cases the derivative in Eq. (11) can be easily estimated or replaced by \( +1 \) or \( -1 \) (i.e., signum \( (d y(k)/d u(k)) \)), this is not the general case. In the indirect adaptive control scheme in Fig. 5, a PE is used to compute the sensitivity of the error function \( J \) with respect to the controller's output. Furthermore, this configuration is particularly useful when the inverse of the plant is ill-defined, i.e., the function \( f(.) \) in Eq. (1) does not admit a true inverse. At the outset, the PE should be off-line trained with a data set sufficiently rich to allow plant identification, and then both the NC and the PE are
on-line trained. For training the controller, the NC and the PE are thought of as, respectively, the variable and fixed parts of a single MNN. The error to be minimized is "backpropagated" through the PE up to the output of the NC, enabling the learning process.

4. Efficient On-Line Training

Slow convergence is the most severe drawback of MNNs. MNNs and BP were originally developed for pattern classification problems, where the training patterns are static, the training procedure and the error function are straightforward, and learning is not undertaken in real time. In control, training patterns (inputs and desired outputs) change with time, several training algorithms and error functions are possible, and real-time learning is necessary.

Although trying to improve the BP algorithm itself is a natural approach toward faster convergence, the idea presented here is based on a different approach, namely, the distinction between sampling and learning frequencies.

In discrete-time control systems, the sampling period $T_s$ is usually chosen via rules-of-thumb in order to make $2\pi/T_s$ much larger than the largest frequency involved in the continuous-time system. It is normally true that increasing the sampling frequency improves the system's performance, but the noticeable improvement rapidly reaches a limitation. Although $T_s$ defines the basic frequency for the controller's operation, in iterative learning systems the frequency at which learning is performed can be thought of as a different time basis. For most practical cases, $T_s$ is much larger than $T_e$, the time spent in one learning operation (or updating period of all network weights), and the ratio $T_s/T_e$ tends to become higher as faster implementations of MNNs (faster computers, special hardware, and so forth) become available. Therefore, if the only concern is time, many learning operations can be carried out during a sampling period, and the normal or the simplest approach of a single updating per period implies waste of processing time. The problem is how to use the available data and time to perform training of the neuromorphic structures in such a way to improve control performance effectively.

4.1 Emulator

Consider that at instant $k+1$ the current plant output $y(k+1)$, the $p+t-1$ previous values of $y$, and the $q+t$ previous values of $u$ have been stored in memory. Then the $t$ pairs $(x_k(k-i), y(k+1-i))$, for $i=0, 1, \ldots, t-1$ can be used as patterns for training the PE at time $k+1$. For $y(k+1-i)=\mathcal{F}_E[x_k(k-i)]$, one possibility is to minimize the following error function

$$J_t(k) = 0.5 \sum_{i=0}^{t-1} \lambda_i [y(k-i) - y_1(k-i)]^2$$

where $\{\lambda_i\} (1 \leq \lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{t-1} \geq 0)$ is a non-increasing positive sequence whose role is to take into account the effect of "forgetting", or emphasizing the most recent patterns.

Example 1: Assume that $y(10)$ was just read (and, therefore, $y(11)$ is not available), $p=3$, $q=2$, and $t=3$. Also assume that $y(9), y(8), \ldots, y(5)$ and $u(9), u(8), \ldots, u(5)$ are available from the memory. Then the PE input vectors can be arranged as follows:

\[
x_k(7) = [y(7), y(6), y(5), u(7), u(6), u(5)]^T,
\]
\[
x_k(8) = [y(8), y(7), y(6), u(8), u(7), u(6)]^T,
\]
\[
x_k(9) = [y(9), y(8), y(7), u(9), u(8), u(7)]^T.
\]

These input vectors and the values $y(8), y(9)$ and $y(10)$ constitute the training patterns at time $k+1=10$. This procedure is illustrated in Fig. 6, where the notation PE_{k,i} indicates the PE's state during the $k$-th sampling interval, after the $i$-th learning operation, $i=0, 1, \ldots, t-1$. Equivalently, $\mathcal{F}_k^i(.)$ denotes the input-
output mapping performed by PE_{k,t}. Clearly \( \varphi_k(x) = \varphi_{k+t}(x) \).

4.2 Controller

4.2.1 Direct Inverse Control Error Approach

The approach described above has an equivalent to the NC. Consider the direct inverse control configuration (Fig. 3) and assume that at instant \( k+1 \) the current output \( y(k+1) \), the \( p+t \) previous values of \( y \) and \( q+t \) previous values of \( u \) are all stored in memory. Then the \( t \) pairs \( x'_C(k-i), u(k-i) \), \( i=0, ..., t-1 \), for \( x'_C(k)=[y(k+1), ..., y(k-p+1), u(k-1), ..., u(k-q)]^T \), can be used as patterns for training the NC at time \( k+1 \). Writing \( u(k-i) = \varphi_k(x'_C(k-i)) \), for the error function

\[
J_t(k) = 0.5 \sum_{i=0}^{t-1} \lambda_i [u(k-i) - u_1(k-i)]^2 \tag{14}
\]

the corresponding \( \delta \)-term for the \( i \)-th pattern becomes simply

\[
\delta_{k,i} = \lambda_i [u(k-i) - u(k-i)] \tag{15}
\]

Clearly, the training error function defined in Eq. (14) is not based on the error at the output of the plant (control error), but on the error related to the direct synthesis of the inverse plant. Therefore, similarly to what was described in Section 3.1, in the direct inverse control error approach the learning does not improve control performance directly, unless training of the NC is carried out in such a way that good generalization through the control space can be expected. This drawback can be minimized by combining this training method with others which directly minimize the control error\(^5\).

Example 2: Assume that \( y(9) \) was just read, \( p=2 \), and \( q=3 \). Also assume that the values \( y(8), y(7), ..., y(5) \) and \( u(8), u(7), ..., u(3) \) are available from the memory. Denoting the current mapping performed by the NC by \( \varphi_k^{x_C}(.), \) (indicating \( k=9 \), no learning yet), the control input \( u(9) \) can be calculated from the relation \( u(9) = \varphi_k^{x_C} [x_C(9)] \), where

\[
x_C(9) = [r(10), y(9), y(8), u(8), u(7), u(6)]^T.
\]

For the learning based on the direct inverse control error approach, the following vectors are then available:

\[
\begin{align*}
x'_C(6) &= [y(7), y(6), y(5), u(5), u(4), u(3)]^T, \\
x'_C(7) &= [y(8), y(7), y(6), u(6), u(5), u(4)]^T, \\
x'_C(8) &= [y(9), y(8), y(7), u(7), u(6), u(5)]^T.
\end{align*}
\]

These vectors and the input values \( u(6), u(7) \), and \( u(8) \) constitute training patterns available at time \( k = 9 \). However, since this kind of training does not minimize directly the control error, in order to achieve good control performance, one should combine this approach with one of the methods described in sections 3.2 or 3.3. Fig. 7 shows the training procedure in which multiple learning based on the direct inverse control error and learning based on the indirect adaptive control configuration are combined, resulting in 4 learning operations per sampling period. The vector \( x_C(8) \) is given by \([r(9), y(8), y(7), u(7), u(6), u(5)]^T\), the notation NC\(^{x_C}\) denotes the NC's state during the \( k \)-th sampling interval, after the \( i \)-th learning iteration (the corresponding mapping is given by \( \varphi_k^{x_C}(.)) \), and the PE is considered perfectly trained, for the sake of simplicity.

4.2.2 Predicted Output Error Approach

A more complex approach can be derived from the adaptive control configurations. Assume that the \( t \) reference values \( r(k+1-i), i=0,1, ..., t-1 \), are also available at instant \( k+1 \), in addition to \( t+p \) values of \( y \), including \( y(k+1) \), and \( t+q \) previous values of \( u \). From Eqs. (6) and (7), this is equivalent to the condition that \( t \) input vectors \( x_C(k-i) \) are available from the memory. According to Eq. (5), at instant \( k-i \), the control input \( u(k-i) \) was generated by

\[
u(k-i) = \varphi_{k-i}^{x_C} [x_C(k-i)] \tag{16}
\]

However, at time \( k+1 \), the stored input vector \( x_C(k-i) \) would yield the virtual control input

\[
u^*(k-i) = \varphi_{k+1-i}^{x_C} [x_C(k-i)] \tag{17}
\]

The corresponding plant response can be predicted from the emulator, as shown in Fig. 8, by

\[
y^*(k+1-i) = \varphi_{k+1}^{x_C} [x_C^*(k-i)] \tag{18}
\]

Fig. 7 Example 2: multiple training of the NC by the direct inverse control error and indirect adaptive control approaches
where the second superscript index of $\varphi(.)$ was omitted for simplicity, and the vectors $x_E^j(k-i)$ are given by

$$x_E^j(k-i)=[y(k-i), \ldots, y(k-p+1-i), u^*(k-2), \ldots, u^*(k-q-i)]^T \quad (19)$$

The NC training can be understood when considering the NC and the PE as a single MNN, in such a way that at time $k+1$, for each input vector $x_C(k-i)$ there is a corresponding predicted error $r(k+1-i) - y^*(k+1-i)$. A possible error function would be

$$J_t(k)=0.5 \sum_{i=0}^{p+q} \lambda_i [r(k-i) - y^*(k-i)]^2 \quad (20)$$

Example 3: Assume that $y(9)$ was just read, $p=2$, and $q=3$. Also assume that $y(8), y(7), \ldots, y(5)$ and $u(8), u(7), \ldots, u(3)$ are available from the memory, as well as $r(9), r(8),$ and $r(7)$. Therefore, the NC input vectors used to compute the 3 most recent control input values are available as follows:

$$x_C(6)=[r(7), y(6), y(5), u(5), u(4), u(3)]^T,$$

$$x_C(7)=[r(8), y(7), y(6), u(6), u(5), u(4)]^T,$$

$$x_C(8)=[r(9), y(8), y(7), u(7), u(6), u(5)]^T.$$  

At time $k+1=9$, a possible training procedure is shown in Fig. 9, where 3 learning operations take place during a sampling period. The predicted plant output values are computed following the scheme in Fig. 8, enabling the utilization of $x_C(6)$ and $x_C(7)$ at time $k+1=9$, while the most recent vector $x_C(8)$ is used in the conventional way.

5. Simulation Results

In order to compare the efficiency of common and proposed on-line training methods, simulation results for a simple plant are illustrated. Consider the temperature control system

$$y(k+1)=Ay(k)+Bu(k)+(1-A)Y_0 \quad (21)$$

where $y(k)$ is temperature in °C, $0 \leq u(k) \leq 5$ and $Y_0$ is the room temperature. In the following, $A=0.997$, $B=0.260$, and $Y_0=25.0$. The model in Eq. (21) and the parameter values for simulation were obtained from a real 7-liter water bath for sampling period $T_s=30$ s.

Comparing Eqs. (21) and (1), it is clear that $P=1$ and $Q=0$.

The indirect adaptive control configuration shown in Fig. 5 was used to control the plant described in Eq. (21). Simple MNNs were chosen for the PE and NC, and convergence was obtained for $p$ ranging from 1 to 3, and $q$ from 0 to 2. The networks were updated according to the general rule

$$\Delta w^n = w^n+1 - w^n = -\eta \frac{\partial J^n}{\partial w^n} + \alpha \Delta w^{n-1} \quad (22)$$

where the superscript indices denote learning iterations, $w$ is a generic weight, $J$ is the error function, $\eta > 0$ and $\alpha \geq 0$ are the learning rate and momentum, respectively.

In the results presented here, $p=1$, $q=0$ (true values), and feedforward MNNs, fully connected between consecutive layers, were used for both the PE and the NC. For simplicity, each MNN had a single hidden layer with 15 neurons, each neuron with a bias and sigmoidal input-output function expressed by $f(z)=[1+\exp(-z+\theta)]^{-1}$, where $z$ is the weighted sum of the inputs to a neuron and $\theta$ is its variable bias. Before the outset, the PE was roughly trained with off-line data, whereas the NC's weights were randomly initialized. From the initial condition $y(0) = Y_0$, the target is to follow a control reference set to 35.0°C for $1 \leq k \leq 60$, 55.0°C for $61 \leq k \leq 120$, and 75.0°C for $121 \leq k \leq 180$. Each simulation cycle, for $k$ ranging from 1 to 180, is called a trial. Therefore, one simulation trial corresponds to $180 \times T_s = 90$ min in the actual control system. After a trial, the weights of both the PE and NC are conserved and a new trial.
starts. Fig. 10 shows the reference, and the control input and output after good convergence was achieved.

The remaining figures compare the performance of NCs carrying out a single learning operation per sampling period with the multiple-learning case. The graphs show the total squared error per trial as a function of the number of trials. In Fig. 11 the NC is initialized in such a way so as to result in small error from the beginning. In Fig. 11(a), training is performed once a period by using the indirect adaptive control configuration, whereas in the lower graphs (Figs. 11(b) and 11(c)) the NC is updated 5 and 10 additional times per period, respectively, by using the direct inverse control error approach. The proposed method improved performance as expected, since the NC is assumed to be relatively well trained from the outset and generalization is, thus, reliable. This fact suggests that this training method can be used for small adjustments of the NC near a good operating point. In Fig. 11(d), each simulation trial with usual on-line training (one learning operation per sampling period) was followed by off-line training based on plant input-output data obtained during the trial, under the condition that the total number of learning operations was the same as in Fig. 11(c). Although the number of training operations was identical in Figs. 11(c) and 11(d), some essential differences must be pointed out. At first, the proposed training method is performed totally on-line, making efficient use of available processing time during each sampling period, and no off-line training is involved. Moreover, since more intense training is performed during the control trial itself, better control performance is expected. Finally, in the on-line training approach, training data is generated continuously, whereas in the off-line case, once a trial is carried out, training data becomes fixed, increasing the possibility of getting trapped in a local minimum. Switching abruptly on- and off-line training schemes based on different approaches may result in large incursions in the NC's weight space, which may cause error increase, as shown in Fig. 11(d).

The performance of a randomly initialized NC executing one learning operation per sampling period via the indirect adaptive control configuration is shown in Fig. 12(a). In Figs. 12(b) and (c), respectively, 5 or 10 additional learning operations based on the proposed predicted output error approach were included. It can be seen that only a few additional learning operations per sampling period resulted in a marked acceleration in convergence speed, significantly reducing the total error. In Fig. 12(d) each simulation trial with usual on-line training (one learning operation per sampling period) was followed by off-line training, under the condition that the total number of learning operations was the same as in Fig. 12(c). It is clear that the convergence improvement achieved by the proposed predicted output error approach was not only due to the increase in the number of learning operations, but also to the training method itself.

It is interesting to note that, although the direct inverse control error approach does accelerate the convergence when the controller is already well
trained, the same does not happen when training is still in the earliest stage. This is shown in Fig. 13, where the NC is randomly initialized, so that the training error is large at the outset. In Fig. 13(a) only conventional on-line training (one learning operation per sampling period) is performed, whereas in Figs. 13(b) and (c) 10 additional learning operations based on the predicted output error and the direct inverse control error approaches, respectively, are included. In the case illustrated, the direct inverse control error approach is even worse than the conventional training. Such a poor performance comes from the fact that, in the proposed direct inverse control error approach, during each sampling period two different training perspectives are undertaken, namely, one based on the direct synthesis of the inverse plant and other based on minimization of the control error. When the NC is not well trained yet, switching these two training perspectives forth and back may cause large changes in the NC's weights and result in slow convergence. On the other hand, when the NC is already well trained, good generalization is expected, and the two training perspectives in the direct inverse control error approach lead to direct reduction of the control error.

The time spent for updating the neuromorphic structures is roughly proportional to the number of learning operations, and basically depend on the structure and number of weights of the MNN being considered. In the simulation results presented here, 11 learning operations of a 3-layer NC with 60 weights took about $11 \times T_s = 300 \text{ms}$ in a personal computer.

6. Conclusion

New on-line training methods for multilayer NCs were proposed. The basic idea is to perform several training operations during each single sampling period in such a way that learning is accelerated. Two different approaches were proposed, namely, the direct inverse control error and the predicted output error approaches.

The direct inverse control error approach is based on the direct synthesis of the inverse plant. Good results can be expected when the NC has been already reasonably trained, so that the NC is able to generalize well through the region of interest in the control space. Conversely, when not enough training has been performed, the direct inverse control error approach may not improve convergence, as shown in the simulation results.

In the predicted output error approach, the error function to be minimized is directly based on the error at the output of the plant, and great convergence speedup can be achieved, even when the NC starts from a random state. In a simulation example, the inclusion of 10 training operations per sampling period, based on the predicted output error approach, resulted in more than five-fold reduction in convergence time and twelve-fold reduction in total squared error.
Starting from a randomly initialized NC, the
presented results suggest that the predicted output
error approach can be used to accelerate learning
until the NC approximates the inverse of the plant,
and then the direct inverse control error approach can
be applied for small adjustments. Hybrid methods
combining the proposed approaches are under study.
Open problems include the selection of optimal values
for the backpropagation learning parameters, as well
as the internal structure of the MNNs used in the
control system.

References
1) K. Funahashi: On the Approximate Realization of
Continuous Mappings by Neural Networks, Neural
Networks, 2, 183/192 (1989)
2) J. Tanomaru and S. Omatu: A Feedforward Back-
propagation Neural Controller, Proc. MTNS'91, 2,
457/462, Kobe (1991)
3) D.E. Rumelhart, G.E. Hinton and R.J. Williams:
Learning Internal Representation by Error Propaga-
tion, in Parallel Distributed Processing, 1, 318/362,
MIT Press (1986)
4) K.S. Narendra and K. Parthasarathy: Identification
and Control of Dynamical Systems Using Neural
Networks, IEEE Transactions on Neural Networks, 1
-1, 4/27 (1990)
5) J. Tanomaru and S. Omatu: Towards Effective Neu-
romorphic Controllers, Proc. IECON'91, 1396/1400,
Kobe (1991)
6) D. Psaltis, A. Sideris and A.A. Yamamura: A Multi-
layered Neural Network Controller, IEEE Control

Julio Tanomaru (Member)

Received the M.E. degree in Information Science from the Univ. of Toku-
shima in 1992, where he is presently a Research Associate. Research interests
include neural networks with applications to adaptive control and optimization.

Sigeru Omatu (Member)

Received the Ph.D. degree in Electronics Engineering from the Univ. of Osaka Prefecture in 1974. Professor at the
Univ. of Tokushima since 1988. Research interests center on neural net-
works, adaptive control, and distributed parameter system theory.