A NEURAL CONTROLLER

Marco Saerens & Alain Soetens

Université Libre de Bruxelles, Belgium

INTRODUCTION

The basic objective of a controller is to provide the appropriate input parameters to a plant in order to obtain the desired output. In control theory, the physical process is referred to as a plant. Given the current state of the plant, the controller has to provide a set of control parameters allowing to reach a given target. It would be interesting if the controller could learn to control the plant in an intuitive and autonomous way by observing the behaviour of the plant and by continuous adaptation to the task. Some interesting progresses have been made in this field by using neural nets. Kuperstein (1) and Kuperstein & Rubinstein (2) use the back-propagation algorithm (Le Cun (3); Rumelhart, Hinton & Williams (4)) to learn sensory-motor coordination from experience. During a training stage, the controller uses the plant as a simulator in order to learn its behaviour. After the learning stage, the neural controller is able to guide the arm to any position in 3D space. Pibon & Gosse 4 5 use the same algorithm for learning coordinated motion of an autonomous robot (see also Mel (6)). Psaltis, Sideris & Yamamura (7) call this kind of learning "general learning". They propose another type of learning algorithm called "specialized learning". "Specialized learning" allows the neural controller to learn in an on-line and autonomous way.

Our goal is first to propose a specialized learning scheme and to study its limitations. Second, we try to show the usefulness of the algorithm by using it in several simulated problems dealing with dynamic processes (dynamic control).

GENERAL LEARNING AND SPECIALIZED LEARNING

In order to learn the characteristics of the plant and to adapt to changing processes, the neural controller must be told what the values of its outputs (the control parameters) should have been. In general, this information is not available: only the error at the outputs of the plant is available. We briefly describe two learning schemes that avoid this problem.

General learning: In the case of general learning, the training stage consists in using the plant to produce a set of input-output pairs. These pairs are then used as patterns for training the neural network. Let the $l_i$ be the control parameters of the plant and the $y_i$ be the outputs of the plant. Let the $y_i$ (underlined) be the desired target (in this case, the desired outputs of the plant). The task of the neural controller is to learn to supply, as its output, the appropriate control parameters ($y_i$) for desired targets ($y_i$), given as input. In other words, the network has to learn to invert the process. During the training stage, the input parameters $l_i$ are chosen randomly within their working range. Those parameters are then injected into the plant which supplies output values $y_i$. Finally, the $y_i$ are used as an input pattern for the network. The back-propagation algorithm is used to train the network to relate the output $l_i$ to the input $y_i$. After this learning stage, the controller is able to provide the correct control parameters to reach any desired target. Psaltis, Sideris & Yamamura (7) point out several drawbacks of this method: first, during the learning stage, the controller is not operational; second, the network cannot limit its working range to the $y_i$ that are actually relevant; and last, the method assumes static targets.

Specialised learning: Psaltis, Sideris & Yamamura (7) propose another learning scheme called specialized learning. Specialized learning differs from general learning by the fact that the controller learns no longer from input-output pairs but from a direct evaluation of the network accuracy with respect to the output of the plant. The network uses the difference between the actual output of the plant $y_i$ and the desired output $y_i$ to change the weight of connections. Specialized learning avoids several drawbacks of general learning: there is no longer a specific training stage during which the network is not operational, and the network learns directly on the domain of relevant $y_i$. Moreover, the network learns continually and can therefore be used with processes having time varying characteristics. Yet, the evaluation of the error from the output requires prior knowledge of the plant. Psaltis, Sideris & Yamamura (7) consider that the plant can be thought of as an additional, though unmodifiable, layer of the neural controller. The weights of the connections leading to this layer are fixed to the values $\frac{\partial y_i}{\partial l_i}$. They propose a modification of the back-propagation algorithm in order to take this layer into account. They compute the errors $e_i$ at the output layer by:

$$e_i = f'(net_i) \sum_j \frac{\partial y_i}{\partial l_j} (\mu_j - y_i)$$

which is similar to what we obtain in equation (1) (see below). The problem is that the "sensitivity derivatives" $\frac{\partial y_i}{\partial l_i}$ can be unknown and difficult to determine. Elsley (8) also proposes a specialized learning algorithm based on back-propagation. In this case, the network outputs the inverse Jacobian of the plant. This

---

1 The following text presents research results of the Belgian National Incentive-program for fundamental research in artificial intelligence initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The scientific responsibility is assumed by the authors.
results in a more complex algorithm showing interesting results: the network is able to learn to control a 2D arm within a control loop. The performances still have to be evaluated on more complex systems and with full dynamic control.

The aim of our work is first to define a specialized learning method based on minimal qualitative knowledge of the plant and, second, to use the proposed algorithm in several problems dealing with dynamic control.

BACK-PROPAGATION THROUGH THE PLANT

In the back-propagation algorithm, the error at the output of the network can be written as follows (Rumelhart, Hinton & Williams (4)):

\[ E = \frac{1}{2} \sum_{q} (\lambda_q - \mu_q)^2 \]

In the case of a neural controller (cf. Figure 1), the final layer provides the inputs \( \lambda_q \) of the plant, which outputs \( \mu_i \) and not the desired target \( \mu_i \). In order to obtain \( \mu_i \), the controller should supply \( \lambda_q \) – not known – as the input of the plant.

![Figure 1: Learning architecture.](image)

The main problem is the evaluation of \( \lambda_q - \mu_q \), since the appropriate inputs \( \lambda_q \) are not known. We recently proposed to perform a first order expansion of the control parameters in terms of the outputs of the plant (Saerens & Soquet (9)). However, a simpler way to solve the problem is to compute the error in terms of the outputs of the plant:

\[ E' = \frac{1}{2} \sum_{q} (\mu_q - \mu_0)^2 \]

where the desired outputs \( \mu_0 \) are known. The back-propagation algorithm implements a gradient descent in \( E'(\mu_0) \):

\[ w_{ij}^{n+1} = w_{ij}^n - \eta \frac{\partial E'}{\partial w_{ij}} \]

To process \( \frac{\partial E'}{\partial w_{ij}} \), we use the chain rule:

\[ \frac{\partial E'}{\partial w_{ij}} = \sum_k \frac{\partial E'}{\partial \mu_k} \frac{\partial \mu_k}{\partial w_{ij}} \]

Let us compute \( \frac{\partial E'}{\partial \mu_k} \), which corresponds to a gradient descent in \( E'(\mu_k) \):

\[ \frac{\partial E'}{\partial \mu_k} = \sum_q \frac{\partial E'}{\partial \mu_q} \frac{\partial \mu_q}{\partial \mu_k} \]

\[ = \sum_q (\mu_q - \mu_0) \frac{\partial \mu_q}{\partial \mu_k} \]

(1)

which is the same result as Psaltis, Sideris & Yamamura (see above). However, the sensitivity derivatives \( \frac{\partial \mu_q}{\partial \mu_k} \) are not known.

Nevertheless, we can approximate them by their sign, which is generally known when qualitative knowledge is available, i.e., when we know the orientation in which the control parameters influence the outputs of the plant:

\[ \frac{\partial \mu_q}{\partial \mu_k} \approx \sum_q (\mu_q - \mu_0) \text{ sign} \left( \frac{\partial \mu_q}{\partial \mu_k} \right) \]

where \( \frac{\partial \mu_q}{\partial \mu_k} \) represents a gradient approximation by sign. Finally, the weight \( w_{ij} \) of the connection between unit \( i \) and unit \( j \) is modified by:

\[ \Delta w_{ij} = -\eta \frac{\partial E'}{\partial w_{ij}} = \eta s_j q_i \]

where \( \eta \) is the learning rate, \( q_i \) the activation of unit \( i \) and \( s_j \) the back-propagated error. If unit \( j \) is an output unit, we find that \( s_j \) is given by:

\[ s_j = f' \left( \text{net}_j \right) \sum_q \text{ sign} \left( \frac{\partial \mu_q}{\partial \mu_k} \right) (\mu_q - \mu_0) \]

(2)

where \( f' \) is the derivative of a logistic activation function and \( \text{net}_j \) is the net total input of unit \( j \). For hidden units, we have the standard rule:

\[ s_j = f' \left( \text{net}_j \right) \sum_k w_{jk} \]

If the \( \lambda_j \) are well-chosen control parameters, they influence the outputs monotonically in the relevant working range. Moreover, if we have some qualitative knowledge of the plant, we can determine the way in which the control parameters \( \lambda_j \) influence the outputs \( \mu_i \) of the plant. In other words, we know the signs of the \( \frac{\partial \mu_i}{\partial \lambda_j} \). Let us take a simple example: the regulation of the temperature and the flow of a shower. The control parameters are the settings of two taps, one for the cold water tap (\( \Omega_1 \)) and one for the hot water tap (\( \Omega_2 \)). The outputs of the shower are the temperature \( T \) and the flow \( P \). Qualitative knowledge is very easy to express: if the setting of the cold water tap grows, temperature decreases \( \frac{dT}{d\Omega_1} < 0 \), etc. In this way, we evaluate the error \( s_j \) using equation (2), by replacing the partial derivatives by...
their sign. This approximation is used for back-propagation. Thus, the neural controller learns in terms of the output of the plant. This approximation performs error minimization in the following sense: if we consider only one output, the scalar product between true gradient descent \( \frac{\partial E'}{\partial k} \) and gradient approximation \( \frac{\partial E'}{\partial k} \) is always positive, which ensures an error decrease.

We propose the following algorithm: (i) The controller receives the actual output state \(-u_i\) of the plant and the desired output parameters \(-u_d\) that have to be provided by the plant. (ii) The network outputs control parameters \(\lambda_i\) associated with \(u_i\). (iii) Those parameters \(\lambda_i\) are input to the plant at time \(t\). (iv) The plant outputs \(u_t = u_d\) at time \((t+\delta t)\). (v) The error \(\delta j\) is evaluated with equation (2) and back-propagated into the network.

This approach can be related to classical adaptive control approaches such as the Model-Reference Adaptive Systems (MRAS) and the Self-Tuning Regulators (STR) (see, for instance, Åström & Wittenmark (11)). In the case of MRAS, the control parameters are modified according to the direction of the gradient of the quadratic error \(E(\lambda)\).

In the working ranges where control parameters influence the outputs of the plant monotonically, we can define a simple learning process based on qualitative knowledge of the plant. The resulting learning algorithm shows some interesting properties: (i) There is no specific learning stage; the controller is self-tuning. (ii) For static targets, the controller is immediately operational but needs several steps to reach the target. The number of steps decreases gradually as learning goes on. (iii) The controller is able to perform autonomous on-line learning on dynamic targets. (iv) The controller can specifically learn in the range of relevant outputs. This provides an autonomous learning based on the back-propagation algorithm: the controller learns "by doing" and not "by examples".

However, the proposed learning scheme can result in undesirable characteristics like slower training; indeed, the algorithm does not perform true gradient descent. In order to evaluate the performance of the algorithm, we have carried out two different simulations that concern kinematic problems with static target (Særensen & Soquet (9), (10)). For the proposed learning scheme, the results showed that: (i) The controller is immediately operational but needs several steps to reach the target. The number of steps converges gradually towards 1.0 which means that the network learns to supply the correct control parameters. After each step the network adjusts its weights by performing back-propagation. (ii) For these two problems, the proposed learning scheme encounters no decrease in performance in comparison with classical learning using appropriate patterns.

**RESULTS**

Three different simulations investigate the possibility of on-line learning on dynamic targets. All the problems satisfy the conditions of applicability of the proposed specialized learning method: qualitative knowledge is available. The neural controller is a network with four layers (two hidden layers). Every unit of each layer is connected with units of adjacent layers. No effort was made to reduce the dynamic range of the inputs and the outputs: they were scaled to reflect continuous values from -1.0 to +1.0, representing normalized signals. We use a back-propagation algorithm with parameters fixed to \(\eta = 0.1\) (the learning rate) and \(\alpha = 0.2\) (the momentum term).

**Arm control in 2D space**

The first experiment involves a 2D arm which has to follow a moving target. The control parameters of the arm are the two angles \(\alpha\) and \(\beta\) (cf. Figure 2), with \(0 < \alpha < 60\) degrees and \(0 < \beta < 90\) degrees. A camera transmits both the coordinates of the tip of the arm \((\alpha, \beta)\) and the object's position \((x, y)\) to the controller. The length of the stems is respectively 1.0 m and 0.6 m. The length of each side of the square in which the target moves is 0.6 m.

![Figure 2: Arm with two degrees of freedom (\(\alpha, \beta\)) following a moving target in a confined space.](image)

The controller has to supply angles that permit the arm to reach the target. Qualitative relations are easy to find: \(\frac{\partial x}{\partial \alpha} < 0\), \(\frac{\partial y}{\partial \alpha} < 0\); etc. Training is performed while the target is moving. The reaction time of the controller is 0.1 s, during which the target moves. The neural controller has to learn on "noisy data" because, when it evaluates the error \(\delta j\), the target has moved from the position the arm had to reach.

During a first series of simulations, the neural network was given the coordinates of the target alone. The controller was, in that case, unable to anticipate the movement of the target because it has no idea of its speed and direction. This is shown in the upper graph of Figure 3, for three different velocities of the target. We observe that the average distance between the arm and the target converges towards reaction time of the arm times velocity of the target, which is the best the controller can do (it learns to move to the previous position of the target).

During a second series of simulations, the neural network was given the difference between previous position and actual position
of the target in addition to actual position. In this case, the network learns to anticipate the movement of the target, as can be seen on the lower graph of Figure 3.

![Graph showing error in position vs. time.](image)

Figure 3: Results for the 2D arm. Each curve shows the averaged results of 5 successive runs, carried out with the same parameter values. Both graphs represent the average distance between the arm and the target in terms of elapsed time for three different velocities of the target.

**Pole-balancing problem**

The second problem is the pole-balancing problem. Figure 4 shows a representation of the pole-balancing problem as defined by Barto, Sutton & Anderson (12). It is a cart to which a rigid pole is hinged. The pole is free to move only in the vertical plane of the cart and track. The controller can only apply forces of -10, 0 or +10 N to the cart. The network is given the angle $\theta$, the rate of change of the angle $\dot{\theta}$, the position $x$ of the cart on the track and the cart velocity $v$. We assume that the equations of motion of the cart-pole system are not known by the controller. We consider that the controller fails when $|l\dot{\theta}| > 45$ degrees or $|l\dot{x}| > 2.4$ m. If the pole is maintained for more than 15 minutes, we consider that the controller has learned its task. The equations of motion of the cart-pole system (see Barto, Sutton & Anderson (12) for details) are integrated by a Runge-Kutta method with a time step of 0.001 s. The parameters specifying mass of cart, mass of pole, pole length, coefficient of friction of cart on track and coefficient of friction of pole on cart are the same as those used by Barto, Sutton & Anderson (12). Qualitative knowledge is trivial to express: $\frac{\partial \theta}{\partial F} < 0$. Tolat & Widrow (13) and Guez & Selinsky (14) have also studied the problem of balancing a pendulum using an adaptive network but, in each case, training was supervised. In both cases, the network was able to keep the pole balanced.

The training algorithm can be described as follows: (i) The network receives the actual state variables ($\theta, \dot{\theta}, x, v$). (ii) The controller back-propagates the error computed by equation (2) and outputs a force associated to the state variables. During these operations, i.e. transmission of information and processing (50 ms), the pole is free falling. (iii) The force is applied during 50 ms to the cart. (iv) State variables are observed and return to (i).

![Graph showing time until failure vs. trials.](image)

Figure 5: Results for the pole-balancing problem.

Figure 5 shows the results of ten successive runs, carried out with the same parameter values. It represents time until failure in terms of trials. After several trials, the controller is able to balance the pole during more than 15 minutes. However, it is difficult to avoid that the pole hits the track boundary ($\pm 2.4$ m). This problem was solved in the following way: if the cart exceeds a right (respectively left) limit or has too much positive (negative) velocity, the controller is asked to maintain the pole at angle $-\theta_0$ ($\theta_0$), which results in the cart being brought into the limit.

**Tracking a moving object**

The last problem deals with tracking a moving object with a camera. The purpose of the control system is to keep a moving object in the centre of the camera screen. The control parameters are the angles $\delta \theta$ and $\delta \theta$ defining horizontal and vertical rotation of the camera around its focal point (cf. Figure 6). The inputs of

![Diagram of a camera with two degrees of freedom.](image)

Figure 6: Camera with two degrees of freedom ($\theta, \phi$) following a moving target ($F$).
the network are the coordinates of the object on the camera screen. The field of vision of the camera is 32 degrees. The motors acting on the camera can perform rotations of a maximum of 20 degrees in one step. The total space covered by the camera is ± 90 degrees for θ and ± 45 degrees for φ. We consider that motor control takes 0.5 s, during which the object is moving. The trajectories of the targets are randomly generated in all possible directions. The velocities are also randomly generated from 10 degrees/s up to 20 degrees/s.

When the target enters the field of vision of the camera, it appears on the screen at P. The coordinates of P are transmitted to the network, which supplies relative rotations θ̇ and φ̇. The camera rotates in 0.5 s during which the target is moving. The new coordinates are then transmitted to the network which computes the error to be back-propagated and supply new angles, etc.

![Error rate vs Trials graph](image)

Figure 7: Results for the camera.

Figure 7 shows the results of ten successive runs. It represents the percentage of failures (percentage of cases in which the target is lost by the camera) in terms of trials. We observe that this percentage gradually decreases towards zero.

CONCLUSION

We have presented a method for the evaluation of the error to be back-propagated. The method allows a neural net to learn to control a plant in an autonomous way, without a specific learning stage. This evaluation is based on qualitative knowledge of the plant. The field of application of the method is specified. We applied the proposed method to three different problems. These three simulations investigate the possibility of on-line learning with dynamic targets. The results show that dynamic control based on the proposed learning scheme is possible for a neural network.

Future work aims to investigate the robustness of the neural controller with respect to changes in the physical parameters and to compare between the neural controller and adaptive control systems such as Model-Reference Adaptive Systems and Self-Tuning Regulators.

REFERENCES

First IEE International Conference on ARTIFICIAL NEURAL NETWORKS

Conference Publication Number 313