CHAPTER 16

# Neural Networks Applications

## 16.1 INTRODUCTION

A list of some applications mentioned in the literature follows:

1. **Aerospace:** High performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, aircraft component fault detection.
2. **Automotive:** Automobile automatic guidance system, warranty activity analysis.
3. **Banking:** Check and other document reading, credit application evaluation.
4. **Credit Card Activity Checking:** Neural networks are used to spot unusual credit card activity that might possibly be associated with loss of a credit card.
5. **Defense:** Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification.
6. **Electronics:** Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling.
7. **Entertainment:** Animation, special effects, market forecasting.
8. **Financial:** Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit-line use analysis, portfolio trading program, corporate financial analysis, currency price prediction.
9. **Industrial:** Neural networks are being trained to predict the output gasses of furnaces and other industrial processes. They then replace complex and costly equipment used for this purpose in the past.
10. **Insurance:** Policy application evaluation, product optimization.
11. **Manufacturing:** Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer-chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process system.

12. **Medical:** Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency-room test advisement.
13. **Oil and Gas:** Exploration.
14. **Robotics:** Trajectory control, forklift robot, manipulator controllers, vision systems.
15. **Speech:** Speech recognition, speech compression, vowel classification, text-to-speech synthesis.
16. **Securities:** Market analysis, automatic bond rating, stock trading advisory systems.
17. **Telecommunications:** Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems.
18. **Transportation:** Truck brake diagnosis systems, vehicle scheduling, routing systems.

## 16.2 ROBOT CONTROL

An important area of application of neural networks is in the field of robotics. Usually, these networks are designed to direct a manipulator, which is the most important form of the industrial robot, to grasp objects, based on sensor data. Another applications include the steering and path-planning of autonomous robot vehicles.

In robotics, the major task involves making movements dependent on sensor data. There are four related problems to be distinguished (Craig, 1989):

- Forward kinematics
- Inverse kinematics
- Dynamics
- Trajectory generation

### 16.2.1 Forward Kinematics

Kinematics is the science of motion, which treats motion without regard to the forces, which cause it. Within this science one studies the position, velocity, acceleration, and all higher order derivatives of the position variables. A very basic problem in the study of mechanical manipulation is that of forward kinematics. This is the static geometrical problem of computing the position and orientation of the end-effector (‘hand’) of the manipulator. Specifically, given a set of joint angles, the forward kinematic problem is to compute the position and orientation of the tool frame relative to the base frame (see Fig. 16.1).

### 16.2.2 Inverse Kinematics

This problem is posed as follows: given the position and orientation of the end-effector of the manipulator, calculate all possible sets of joint angles which could be used to attain this given position and orientation. This is a fundamental problem in the practical use of manipulators.

The inverse kinematic problem is not as simple as the forward one. Because the kinematic equations are nonlinear, their solution is not always easy or even possible in a closed form. Also, the questions of existence of a solution, and of multiple solutions, arise. Solving this problem is a least requirement for most robot control systems.

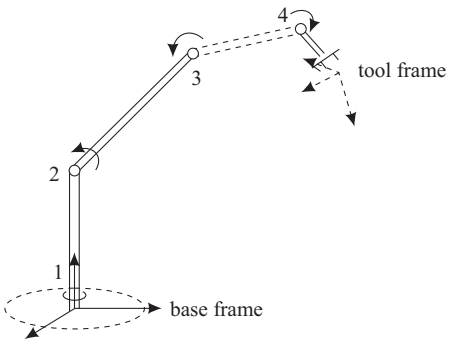


**Fig. 16.1** An exemplar robot manipulator.

### 16.2.3 Dynamics

Dynamics is a field of study devoted to studying the forces required to cause motion. In order to accelerate a manipulator from rest, glide at a constant end-effector velocity, and finally decelerate to a stop, a complex set of torque functions must be applied by the joint actuators. In dynamics not only the geometrical properties (kinematics) are used, but also the physical properties of the robot are taken into account. Take for instance the weight (inertia) of the robot arm, which determines the force required to change the motion of the arm. The dynamics introduces two extra problems to the kinematic problems.

- The robot arm has a 'memory'. Its responds to a control signal depends also on its history (e.g. previous positions, speed, acceleration).
- If a robot grabs an object then the dynamics change but the kinematics don't. This is because the weight of the object has to be added to the weight of the arm (that's why robot arms are so heavy, making the relative weight change very small).

### 16.2.4 Trajectory Generation

To move a manipulator from here to there in a smooth, controlled fashion each joint must be moved via a smooth function of time. Exactly how to compute these motion functions is the problem of trajectory generation.

In the first section of this chapter we will discuss the problems associated with the positioning of the end-effector (in effect, representing the inverse kinematics in combination with sensory transformation).

### 16.2.5 End-Effector Positioning

The final goal in robot manipulator control is often the positioning of the hand or end-effector in order to be able to, e.g., pick up an object. With the accurate robot arm that are manufactured, this task is often relatively simple, involving the following steps:

- Determine the target coordinates relative to the base of the robot. Typically, when this position is not always the same, this is done with a number of fixed cameras or other sensors which observe the work scene, from the image frame determine the position of the object in that frame, and perform a pre-determined coordinate transformation;

- With a precise model of the robot (supplied by the manufacturer), calculate the joint angles to reach the target (i.e., the inverse kinematics). This is a relatively simple problem;
- Move the arm (dynamics control) and close the gripper. Gripper control is not a trivial matter at all, but we will not focus on that.

### 16.2.5a Involvement of Neural Networks

So if these parts are relatively simple to solve with a high accuracy, why involve neural networks? The reason is the applicability of robots. When 'traditional' methods are used to control a robot arm, accurate models of the sensors and manipulators (in some cases with unknown parameters which have to be estimated from the system's behavior; yet still with accurate models as starting point) are required and the system must be calibrated. Also, systems, which suffer from wear-and-tear, need frequent recalibration or parameter determination. Finally, the development of more complex (adaptive!) control methods allows the design and use of more flexible (i.e., less rigid) robot systems, both on the sensory and motor side.

### 16.2.6 Camera-Robot Coordination in Function Approximation

The system we focus on in this section is a work floor observed by fixed cameras, and a robot arm. The visual system must identify the target as well as determine the visual position of the end-effector.
The target position $x^{\text{target}}$ together with the visual position of the hand $x^{\text{hand}}$ are input to the neural controller $N(\bullet)$. This controller then generates a joint position $\theta$ for the robot:

$$\theta = N(x^{\text{target}}, x^{\text{hand}}) \qquad ...(16.1)$$

We can compare the neurally generated $\theta q$ with the optimal $\theta q_0$ generated by a fictitious perfect controller $R(\bullet)$:

$$\theta_0 = R(x^{\text{target}}, x^{\text{hand}}) \qquad ...(16.2)$$

The task of learning is to make the $N$ generate an output 'close enough' to $\theta_0$.

There are two problems associated with teaching $N(\bullet)$:

1. Generating learning samples which are in accordance with eq. (8.2). This is not trivial, since in useful applications $R(\bullet)$ is an unknown function. Instead, a form of self-supervised or unsupervised learning is required. Some examples to solve this problem are given below;
2. Constructing the mapping $N(\bullet)$ from the available learning samples. When the (usually randomly drawn) learning samples are available, a neural network uses these samples to represent the whole input space over which the robot is active. This is evidently a form of interpolation, but has the problem that the input space is of a high dimensionality, and the samples are randomly distributed.

We will discuss three fundamentally different approaches to neural networks for robot end-effector positioning. In each of these approaches, a solution will be found for both the learning sample generation and the function representation.

### 16.2.6a Approach-1: Feed-forward Networks

When using a feed-forward system for controlling the manipulator, a self-supervised learning system must be used. One such a system has been reported by Psaltis, Sideris and Yamamura (1988). Here, the network, which is constrained to two-dimensional positioning of the robot arm, learns by experimentation. Three methods are proposed:

1. **Indirect learning:** In indirect learning, a Cartesian target point $x$ in world coordinates is generated, e.g., by a two cameras looking at an object. This target point is fed into the network, which generates an angle vector $\theta$. The manipulator moves to position $\theta$, and the cameras determine the new position $x'$ of the end-effector in world coordinates. This $x'$ again is input to the network, resulting in $\theta'$. The network is then trained on the error $\varepsilon_1 = \theta - \theta'$ (see Fig. 16.2).

   However, minimization of $\varepsilon_1$ does not guarantee minimization of the overall error $\varepsilon = x - x'$. For example, the network often settles at a 'solution' that maps all $x$'s to a single $\theta$ (i.e., the mapping II).
2. **General learning:** The method is basically very much like supervised learning, but here the plant input $\theta$ must be provided by the user. Thus the network can directly minimize $|\theta - \theta'|$. The success of this method depends on the interpolation capabilities of the network. Correct choice of $\theta$ may pose a problem.

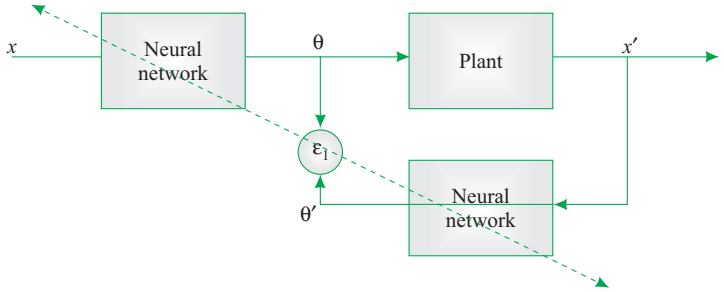


**Fig. 16.2** Indirect learning system for robotics. In each cycle, the network is used in two different places: first in the forward step, then for feeding back the error.

3. **Specialized learning:** Keep in mind that the goal of the training of the network is to minimize the error at the output of the plant: $\in = x - x'$. We can also train the network by 'backpropagating' this error trough the plant (compare this with the backpropagation of the error in Chapter 12). This method requires knowledge of the Jacobian matrix of the plant. A Jacobian matrix of a multidimensional function $F$ is a matrix of partial derivatives of $F$, i.e., the multidimensional form of the derivative. For example, if we have $Y = F(x)$, i.e.,

$$y_1 = f_1(x_1, x_2, ..., x_n)$$
$$y_2 = f_2(x_1, x_2, ..., x_n)$$
$$\vdots$$
$$y_m = f_m(x_1, x_2, ..., x_n)$$

then

$$\delta y_1 = \frac{\partial f_1}{\partial x_1}\delta x_1 + \frac{\partial f_1}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_1}{\partial x_n}\delta x_n$$

$$\delta y_2 = \frac{\partial f_2}{\partial x_1}\delta x_1 + \frac{\partial f_2}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_2}{\partial x_n}\delta x_n$$

$$\vdots$$

$$\delta y_m = \frac{\partial f_m}{\partial x_1}\delta x_1 \frac{\partial f_m}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_m}{\partial x_n}\delta x_n$$

or

$$\delta Y = \frac{\partial F}{\partial X}\delta X \qquad \ldots (16.3)$$

Eq. (16.3) is also written as

$$\delta Y = J(X)\ \delta X \qquad \ldots (16.4)$$

where $J$ is the Jacobian matrix of $F$. so, the Jacobian matrix can be used to calculate the change in the function when its parameters change.

Now, in this case we have

$$J_{ij} = \left[\frac{\partial P_i}{\partial \theta_i}\right] \qquad \ldots(16.5)$$

where $P_i(\theta)$ the ith element of the plant output for input θ. The learning rule applied here regards the plant as an additional and unmodifiable layer in the neural network. The total error $\in = x - x'$is propagated back through the plant by calculating the $\delta_j$:

$$\delta_j = F(s_j)\sum_i \delta_i \frac{\partial P_i(\theta)}{\partial \theta_j} \qquad \ldots(16.6)$$

$$\delta_i = x - x'$$

where $I$ is used to change the scalar $\theta_j$ into a vector. When the plant is an unknown function, $\frac{\partial P_i(\theta)}{\partial \theta_j}$ can be approximated by

$$\frac{\partial P_i(\theta)}{\partial \theta_j} \approx \frac{P_i(\theta + h\theta_j e_j) - P_i(\theta)}{\partial \theta_j} \qquad \ldots(16.7)$$

This approximate derivative can be measured by slightly changing the input the plant and measuring the changes in the output.

A somewhat similar approach is taken in (Krose, Korst, and Groen, 1990) and (Smagt and Krose, 1991). Again a two-layer feed-forward network is trained with back-propagation. However, instead of

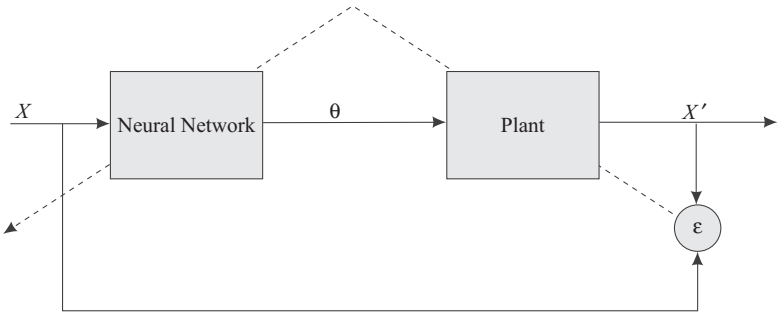


**Fig. 16.3** The system used for specialized learning.

calculating a desired output vector the input vector which should have invoked the current output vector is reconstructed, and back-propagation is applied to this new input vector and the existing output vector.

The configuration used consists of a monocular manipulator, which has to grasp objects. Due to the fact that the camera is situated in the hand of the robot, the task is to move the hand such that the object is in the centre of the image and has some predetermined size (in a later article, a biologically inspired system is proposed (Smagt, Krose, and Groen, 1992) in which the visual flow-field is used to account for the monocularity of the system, such that the dimensions of the object need not to be known anymore to the system).

One step towards the target consists of the following operations:

1. Measure the distance from the current position to the target position in camera domain, *x;*
2. Use this distance, together with the current state θ of the robot, as input for the neural network. The network then generates a joint displacement vector Δθ;
3. Send Δθ to the manipulator;
4. Again measure the distance from the current position to the target position in camera domain, $x'$;
5. Calculate the move made by the manipulator in visual domain, $x - R_i^{l+1} x'$, where $R_i^{l+1}$ is the rotation matrix of the second camera image with respect to the first camera image;
6. Teach the learning pair $(x - R_i^{l+1} x', \theta; \Delta\theta)$ to the network.

This system has shown to learn correct behavior in only tens of iterations, and to be very adaptive to changes in the sensor or manipulator (Smagt & Krose, 1991; Smagt, Groen, & Krose, 1993).

By using a feed-forward network, the available learning samples are approximated by a single, smooth function consisting of a summation of sigmoid functions. A feed-forward network with one layer of sigmoid units is capable of representing practically any function. But how are the optimal weights determined in finite time to obtain this optimal representation?

Experiments have shown that, although a reasonable representation can be obtained in a short period of time, an accurate representation of the function that governs the learning samples is often not feasible or extremely difficult (Jansen et al., 1994). The reason for this is the global character of the approximation obtained with a feed-forward network with sigmoid units: every weight in the network has a global effect on the final approximation that is obtained.

Building local representations is the obvious way out: every part of the network is responsible for a small subspace of the total input space. Thus accuracy is obtained locally (keep it small and simple). This is typically obtained with a Kohonen network.

### 16.2.6b Approach 2: Topology Conserving Maps

Ritter, Martinetz, and Schulten (1989) describe the use of a Kohonen-like network for robot control. We will only describe the kinematics part, since it is the most interesting and straightforward.

The system described by Ritter et al. consists of a robot manipulator with three degrees of freedom (orientation of the end-effector is not included), which has to grab objects in 3D-space. The system is observed by two fixed cameras which output their ($x$, $y$) coordinates of the object and the end effector (see Fig. 16.4).

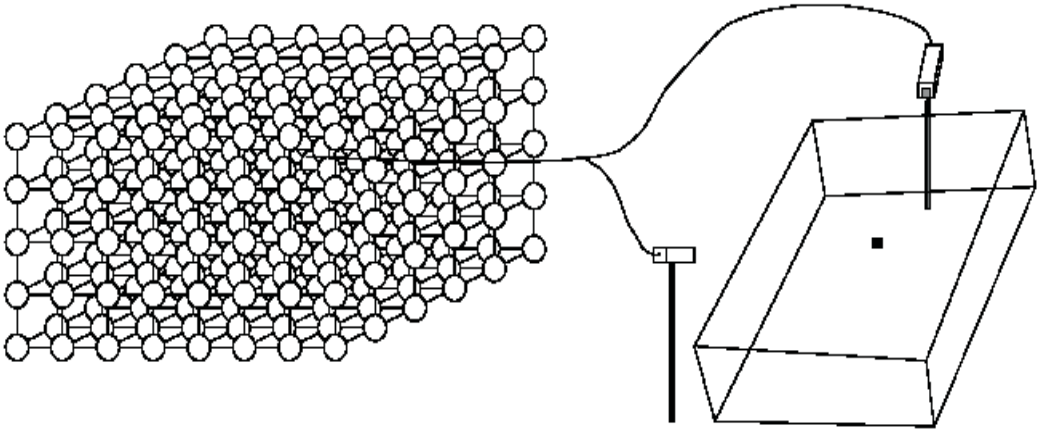

**Fig. 16.4** A Kohonen network merging the output of two cameras.

Each run consists of two movements. In the gross move, the observed location of the object $x$ (a four-component vector) is input to the network. As with the Kohonen network, the neuron $k$ with highest activation value is selected as winner, because its weight vector $w_k$ is nearest to $x$.

The neurons, which are arranged in a 3-dimensional lattice, correspond in a 1–1 fashion with subregions of the 3 D workspace of the robot, i.e., the neuronal lattice is a discrete representation of the workspace. With each neuron a vector $\theta$ and Jacobian matrix $A$ are associated. During gross move $\theta_k$ is fed to the robot which makes its move, resulting in retinal coordinates $x_g$ of the end-effector.

To correct for the discretization of the working space, an additional move is made which is dependent of the distance between the neuron and the object in space $w_k - x$; this small displacement in Cartesian space is translated to an angle change using the Jacobian $A_k$:

$$\theta^{\text{final}} = \theta_k + A_k(x - w_k) \qquad ...(16.8)$$

which is a first-order Taylor expansion of $\theta^{\text{final}}$. The final retinal coordinates of the end-effector after this fine move are in $x_f$.

Learning proceeds as follows: when an improved estimate $(\theta, A)^*$ has been found, the following adaptations are made for all neurons $j$:

$$w_j^{\text{new}} = w_j^{\text{old}} + \gamma(t)\, g_{jk}(t)\, (x - w_j^{\text{old}})$$

$$(\theta, A)_j^{\text{new}} = (\theta, A)_j^{\text{old}} + \gamma'(t)\, g'_{jk}(t)\, ((\theta, A)_j^* - (\theta, A)_j^{\text{old}})$$

If $g_{jk}(t) = g'_{jk}(t) = \delta_{jk}$, this is similar to perceptron learning. Here, as with the Kohonen learning rule, a distance function is used such that $g_{jk}(t)$ and $g'_{jk}(t)$ are Gaussians depending on the distance between neurons $j$ and $k$ with a maximum at $j = k$. An improved estimate $(\theta, A)^*$ is obtained as follows:

$$\theta^* = \theta_k + A_k(x - x_f) \qquad ...(16.9)$$

$$A^* = A_k + A_k(x - w_k - x_f + x_g)\frac{(x_f - x_g)^T}{\|x_f - x_g\|^2} = A_k + (\Delta\theta - A_k\Delta x)\frac{\Delta x^T}{\|\Delta x\|^2} \qquad ...(16.10)$$

In eq. (16.9), the final error $x - x_f$ in Cartesian space is translated to an error in joint space via multiplication by $A_k$. This error is then added to $\theta_k$ to constitute the improved estimate $\theta^*$ (steepest descent minimization of error).

In eq. (16.10), $\Delta x = x_f - f_g$, i.e., the change in retinal coordinates of the end-effector due to the fine movement, and $\Delta\theta = A_k(x - w_k)$, i.e., the related joint angles during fine movement. Thus eq. (16.10) can be recognized as an error-correction rule of the Widrow-Hoff type for Jacobians $A$. It appears that after 6,000 iterations the system approaches correct behavior, and that after 30,000 learning steps no noteworthy deviation is present.

### 16.2.7 Robot Arm Dynamics

While end-effector positioning via sensor{robot coordination is an important problem to solve, the robot itself will not move without dynamic control of its limbs. Again, accurate control with non-adaptive controllers is possible only when accurate models of the robot are available, and the robot is not too susceptible to wear-and-tear. This requirement has led to the current-day robots that are used in many factories. But the application of neural networks in this field changes these requirements.

One of the first neural networks which succeeded in doing dynamic control of a robot arm was presented by Kawato, Furukawa, and Suzuki (1987). They describe a neural network which generates motor commands from a desired trajectory in joint angles. Their system does not include the trajectory generation or the transformation of visual coordinates to body coordinates.

The network is extremely simple. In fact, the system is a feed-forward network, but by carefully choosing the basis functions, the network can be restricted to one learning layer such that finding the optimal is a trivial task. In this case, the basis functions are thus chosen that the function that is approximated is a linear combination of those basis functions.

Dynamics model. The manipulator used consists of three joints as the manipulator in Fig. 16.1 without wrist joint. The desired trajectory $\theta_d(t)$, which is generated by another subsystem, is fed into the inverse-dynamics model (Fig. 16.5). The error between $\theta_d(t)$ and $\theta(t)$ is fed into the neural model.

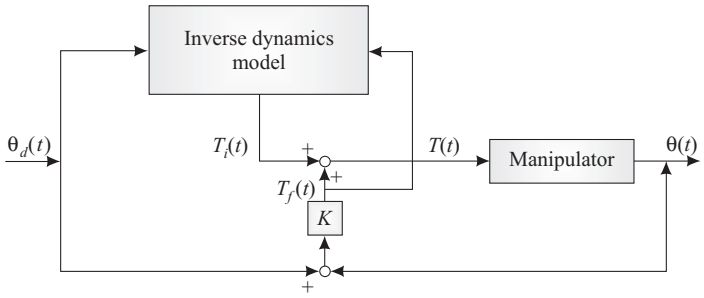


**Fig. 16.5** The neural model proposed by Kawato et al.

The neural model, which is shown in Fig. 16.6, consists of three perceptrons, each one feeding in one joint of the manipulator. The desired trajectory $\theta_d = (\theta_{d1}, \theta_{d2}, \theta_{d3})$ is feed into 13 nonlinear subsystems. The resulting signals are weighted and summed, such that

$$T_{ik}(t) = \sum_{i=1}^{13} w_{lk} x_{lk} \quad (k = 1, 2, 3) \qquad ...(16.11)$$

with

$$x_{l1} = f_1(\theta_{d1}(t), \theta_{d2}(t), \theta_{d3}(t))$$

$$x_{l2} = x_{l3} = g_l(\theta_{d1}(t), \theta_{d2}(t), \theta_{d3}(t))$$

and $f_l$ and $g_l$ as in Table 16.1.

The feedback torque $T_f(t)$ in Fig. 16.5 consists of

$$T_{fk}(t) = K_{pk}(\theta_{dk}(t) = \theta_k(t)) + K_{vk} \frac{d\theta(t)}{dt}, \quad (k\text{–}1,2,3)$$

$$K_{vk} = 0 \text{ unless } |\theta_k(t) - \theta_{dk}(\text{objective point})| < \varepsilon$$

The feedback gains $K_p$ and $K_v$ were computed as $(517.2, 746.0, 191.4)^T$ and $(16.2, 37.2, 8.4)^T$ .

Next, the weights adapt using the delta rule

$$\gamma \frac{dw_{ik}}{dt} = x_{ik} T_1 = x_{ik}(T_{jk} - T_{ik}), \quad (k = 1,2,3) \qquad ...(16.12)$$

A desired move pattern is shown in Fig. 16.7. After 20 minutes of learning the feedback torques are nearly zero such that the system has successfully learned the transformation. Although the applied patterns are very dedicated, training with a repetitive pattern sin $(w_k t)$, with $w_1 : w_2 : w_3 = 1 : \sqrt{2} : \sqrt{3}$ is also successful.

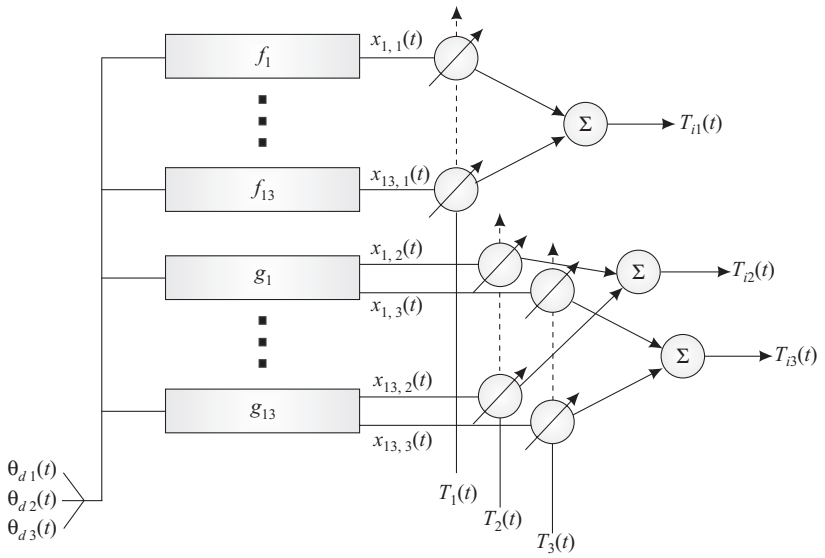


**Fig. 16.6** The neural network used by Kawato et al. There are three neurons, one per joint in the robot arm. Each neuron feeds from thirteen nonlinear subsystems. The upper neuron is connected to the rotary base joint (cf. joint 1 in Figure 16.1), the other two neurons to joints 2 and 3.

**Table 16.1:** Nonlinear transformations used in the Kawato model.

| $l$ | $f_1(\theta_1, \theta_2, \theta_3)$ | $g_1(\theta_1, \theta_2, \theta_3)$ |
|---|---|---|
| 1 | $\ddot{\theta}_1$ | $\ddot{\theta}_2$ |
| 2 | $\ddot{\theta}_1 \sin^2 \theta_2$ | $\ddot{\theta}_3$ |
| 3 | $\ddot{\theta}_1 \cos^2 \theta_2$ | $\ddot{\theta}_2 \cos \theta_3$ |
| 4 | $\ddot{\theta}_1 \sin^2 (\theta_2 + \theta_3)$ | $\ddot{\theta}_3 \cos \theta_3$ |
| 5 | $\ddot{\theta}_1 \cos^2 (\theta_2 + \theta_3)$ | $\dot{\theta}_1^2 \sin \theta_2 \cos \theta_2$ |
| 6 | $\ddot{\theta}_1 \sin \theta_2 \sin (\theta_2 + \theta_3)$ | $\dot{\theta}_1^2 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$ |
| 7 | $\ddot{\theta}_1 \dot{\theta}_2 \sin \theta_2 \cos \theta_2$ | $\dot{\theta}_1^2 \sin \theta_2 \cos (\theta_2 + \theta_3)$ |
| 8 | $\ddot{\theta}_1 \dot{\theta}_2 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$ | $\dot{\theta}_1^2 \cos \theta_2 \sin (\theta_2 + \theta_3)$ |
| 9 | $\ddot{\theta}_1 \dot{\theta}_2 \sin \theta_2 \cos (\theta_2 + \theta_3)$ | $\dot{\theta}_2^2 \sin \theta_3$ |
| 10 | $\ddot{\theta}_1 \dot{\theta}_2 \cos \theta_2 \sin (\theta_2 + \theta_3)$ | $\dot{\theta}_3^2 \sin \theta_3$ |
| 11 | $\ddot{\theta}_1 \dot{\theta}_3 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$ | $\dot{\theta}_2 \ddot{\theta}_3 \sin \theta_3$ |
| 12 | $\ddot{\theta}_1 \dot{\theta}_3 \sin \theta_2 \cos (\theta_2 + \theta_3)$ | $\dot{\theta}_2$ |
| 13 | $\ddot{\theta}_1$ | $\dot{\theta}_3$ |

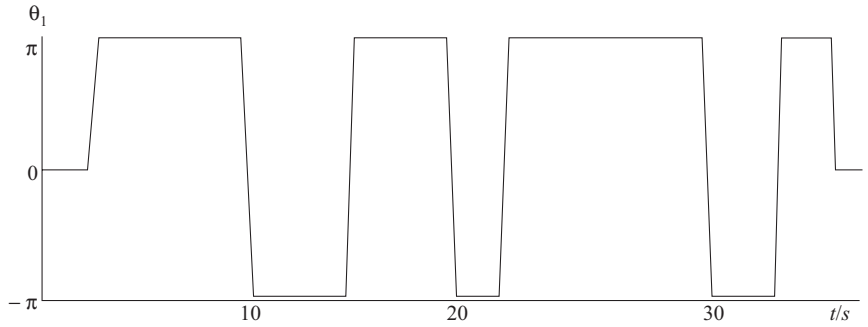


**Fig. 16.7** The desired joint pattern for joint 1. Joints 2 and 3 have similar time patterns.

The usefulness of neural algorithms is demonstrated by the fact that novel robot architectures, which no longer need a very rigid structure to simplify the controller, are now constructed.

## 16.3 DETECTION OF TOOL BREAKAGE IN MILLING OPERATIONS

The recent trend in manufacturing is to achieve integrated and self-adjusting machining systems, which are capable of machining varying parts without the supervision of operators. The absence of human supervision requires on-line monitoring of machining operation, ensuring safe and efficient metal removal rate and taking corrective actions in the event of failures and disturbances (Yusuf, 1998). One of the most important monitoring requirements is a system capable of detecting tool breakages on-line. Unless recognized in time, tool breakage can lead to irreparable damage to the workpiece and possibly to the machine tool itself.

The cutting force variation characteristics of normal and broken tools are different. With the normal and broken tool cutting force variation signals is possible to train neural networks. The milling operations can be monitored with the neural network, after training. The use of adaptive resonance theory (ART) type neural network was evaluated for detections of tool breakage, in this study (Ibrahim and Mclaughlin, 1993). Also simulation-based training is proposed to reduce the cost of preparing the systems that monitor the real cutting signals.

Neural networks with parallel processing capability and robust performances provide a new approach to adaptive pattern recognition. Adaptive Resonance Theory (ART 2) architectures are neural networks that carry out stable self-organization of recognition codes for arbitrary sequence of input patterns. Artificial neural networks refer to a group of architectures of the brain (Cheng and Sheng, 1995). Neural networks are also classified as supervised and unsupervised according to their learning characteristics. In unsupervised learning, the neural network classifies the signals by itself.

In this section, ART type unsupervised neural network paradigm was used for detection of tool breakage. ART paradigm was used for the following reasons.

(a) The training of paradigm is much faster than the back-propagation technique;

(b) The back-propagation technique generalizes the given information in order to store it inside the initially selected hidden layers. The back propagation technique cannot give reliable decisions on the sufficiency of previous training; and

(c) ART has very important advantage since it can be trained in the field and continuously updates previous experience.

The unsupervised ART neural networks can monitor the signal based on previous experience and can update itself automatically while it is monitoring the signals (Carpenter and Grossberg, 1991). When as ART network receives an input pattern, this bottom-up pattern is compared to the top-down, or already known patterns. If the input pattern is matched with known pattern in memory, the weights of the model are changed to update the category. If the new pattern cannot be classified in a known category, it is coded and classified as a new category.

Another important issue is the training of the neural network. It is extremely expensive and time consuming to collect cutting force data at different cutting conditions with normal and broken tools. To overcome this problem, simulation-based training of neural networks was introduced. Simulated data was used to select the best vigilance of the ART 2 type neural network and to evaluate the performance of paradigm. The theoretical background of ART 2 type neural network, the proposed data monitoring system and their performance is presented in the paper.

### 16.3.1 Unsupervised Adaptive Resonance Theory (ART) Neural Networks

The theory of adaptive resonance networks was first introduces by Carpenter and Grossberg, (1987). Adaptive resonance occurs when the input to a network and the feed back expectancies match. The ART 2 neural networks developed by Carpenter and Grossberg (1991) self-organize recognition codes in real time.

The basic ART 2 architectures consist of two types of nodes, the short term memory (STM) nodes, which are temporary and flexible, and the long term memory (LTM) nodes; which are permanent and stable. The input pattern ( $i$ ) is received by the STM, where it is normalized, matched, learned, and stored in the LTM ($z_{ji}$). The STM is divided into two sets of nodes, $F_1$ and $F_2$. The STM $F_1$ nodes are used for normalization, control, gain and learning procedures. The $F_1$ field in ART 2 includes a combination of normalization and noise suppression, in addition to the comparison of the bottom-up and top-down signals needed for the reset mechanism. To accomplish this, $F_1$ uses the following equations to calculate the nodes:

$$u_i = \frac{v_i}{e + \|v\|} \qquad ...(16.13)$$

$$w_i = S_i + au_i \qquad ...(16.14)$$

$$p_i = u_i + \Sigma q(y_i) z_{ji} \qquad ...(16.15)$$

$$q_i = \frac{p_i}{e + \|p\|} \qquad ...(16.16)$$

$$v_i = f(x_i) + bf(q_i) \qquad ...(16.17)$$

$$x_i = \frac{w_i}{e + \|w\|} \qquad ...(16.18)$$

Here $\|p\|$, $\|v\|$ and $\|w\|$ denote the norms of the vectors $p$, $v$ and $w$, and $s_i$ is the input. The non-linear signal function in equation (5) is used for noise suppression . the activation function ($f$ ) is given by the equation.

$$f(x) = \begin{cases} x & \text{if } x \geq \theta \\ 0 & \text{if } 0 \leq x \geq \theta \end{cases} \quad ...(16.19)$$

where θ is an appropriate constant. The function $f$ filters the noise from the signal. θ can be set to zero for the case where filtering is not desired. The constants $a$, $b$, and $e$ are selected based on the particular application.

The STM $F_2$ nodes are used for the matching procedure. $F_2$ equations select or activate nodes in the LTM. When $F_2$ chooses a node, all other nodes in the LTM are inhibited, and only one is allowed to interact with the STM. The node that gives the largest sum with the $F_1$, $F_2$ input pattern (bottom-up) is the key property that is used for node selection.

Bottom-up inputs are calculated as in ART 2 [Fauselt, 2004]

$$T_j = \sum_i p_i z_{ji} \quad ...(16.20)$$

The $j^{th}$ node is selected if equation (16.20) is satisfied.

$$T_j = \max \{ T_j : 1, 2 \ldots N\} \quad ...(16.21)$$

Competition on $F_2$ results in contrast enhancement where a single winning node is chosen each time. The output function of $F_2$ is given by

$$g(y_i) = \begin{cases} d & \text{if } T_i : j = 1, 2, \ldots N \\ 0 & \text{otherwise} \end{cases} \quad ...(16.22)$$

Equation (3) takes the following form:

$$p_i = \begin{cases} u_i & \text{if } F_2 \text{ is inactive} \\ u_i + dz_{ij} & \text{if } j\text{th node is } F_2 \text{ is active} \end{cases} \quad ...(16.23)$$

The bottom-up and top-down LTM equations are

bottom-up ($F_1 \rightarrow F_2$) :

$$\frac{dz_{ij}}{dt} = g(y_j)\,[p_i - z_{ij}] \quad ...(16.24)$$

top–bottom ( $F_2 \rightarrow F_1$) :

$$\frac{dz_{ij}}{dt} = g(y_j)\,[p_i - z_{ij}] \quad ...(16.25)$$

When $F_2$ is active, then equations (16.24) and (16.25) are modified from equation (16.22) to:

$$\frac{dz_{ij}}{dt} = d[p_i - z_{ij}] \qquad ...(16.26)$$

where $d$ is a constant ( $0 < d < 1$). An orienting ART 2 sub system is used to decide, if a new pattern can be matched to a known pattern by comparing with a given vigilance parameter, ρ:

$$r_i = \frac{u_i + cp_i}{e + ||u|| + ||cp||} \qquad ...(16.27)$$

If $||r|| < \rho - e$, then $F_2$ resets another node. If $||r|| \geq \rho - e$ , a match has been found and the new pattern is learned by the system. The LTM node weights are recalculated and the pattern is learned by the system. If no match has been found after all nodes have been activated, a new node is created, and the new pattern is stored.

## 16.3.2 Results and Discussion

The experimental data was collected with a fo1ur flute end mill of 12.07 mm diameter at various cutting conditions. The ART neural network monitored the profile of the resultant force in different tests. In the three tests, experiments were done at different feed rates with the good and broken tool. The spindle speed, feed rate, and depth of cut of these different conditions are out lined in Tables 16.2-16.5. The neural network did not have any prior information at the beginning of each test. In each one, the neural network inspected the resultant force profile and placed it into a category or initiated a new category if it was found to be different.

The vigilance of the ART 2 selected either 0.96 or 0.98 in all the tests. The ART assigned 2, 2, 3, 1 and 3 different categories for the good tool. For the broken tool 2, 1, 1, 1 and 3 different categories were selected. In all the tests, the neural network classified the good and broken tools in different categories. As seen in Tables 16.2-16.5, the neural network generated only one category in the $2^{nd}$ (Table 16.3) , $3^{rd}$ (Table16.4) and $4^{th}$ (Table 16.5) tests for the broken tools. On the other hand, the neural network assigned more nodes to the signal of a good tool with offset. It indicates that the broken tool signals are more similar to each other at different cutting conditions compared to the force patterns of normal tools.

**Table-16.2** Classification of experimental data with the ART. Vigilance of the neural network was 0.96. The ART used four categories to classify all of the data.

| *Spindle speed (rpm)* | *Depth of cut (mm)* | *Feed rate mm/min* | *Tool condition* | *Category* |
|---|---|---|---|---|
| 500 | 1.016 | 50.8 | G | 1 |
| 500 | 1.016 | 50.8 | B | 2 |
| 500 | 1.016 | 101.6 | G | 1 |
| 500 | 1.016 | 101.6 | B | 3 |
| 500 | 1.016 | 203.2 | G | 1 |
| 500 | 1.016 | 203.2 | B | 3 |
| 500 | 1.016 | 254 | G | 4 |
| 500 | 1.016 | 254 | B | 5 |

**Table16.3:** Classification of experimental data with the ART. Vigilance of the neural network was 0.96. The ART used three categories to classify all of the data.

| *Spindle speed (rpm)* | *Depth of cut (mm)* | *Feed rate mm/min* | *Tool condition* | *Category* |
|---|---|---|---|---|
| 500 | 1.524 | 50.8 | G | 1 |
| 500 | 1.524 | 50.8 | B | 2 |
| 500 | 1.524 | 101.6 | G | 3 |
| 500 | 1.524 | 101.6 | B | 2 |
| 500 | 1.524 | 203.2 | G | 1 |
| 500 | 1.524 | 203.2 | B | 2 |
| 500 | 1.524 | 254 | G | 3 |
| 500 | 1.524 | 254 | B | 2 |

**Table 16.4:** Classification of experimental data with the ART 2. Vigilance of the neural network was 0.98. The ART 2 used four categories to classify all of the data.

| *Spindle speed (rpm)* | *Depth of cut (mm)* | *Feed rate mm/min* | *Tool condition* | *Category* |
|---|---|---|---|---|
| 700 | 1.016 | 50.8 | G | 1 |
| 700 | 1.016 | 50.8 | B | 2 |
| 700 | 1.016 | 101.6 | G | 3 |
| 700 | 1.016 | 101.6 | B | 2 |
| 700 | 1.016 | 203.2 | G | 4 |
| 700 | 1.016 | 203.2 | B | 2 |
| 700 | 1.016 | 254 | G | 4 |
| 700 | 1.016 | 254 | B | 2 |

**Table 16.5:** Classification of experimental data with the ART 2. Vigilance of the neural network was 0.96. The ART 2 used two categories to classify all of the data.

| *Spindle speed (rpm)* | *Depth of cut (mm)* | *Feed rate mm/min* | *Tool condition* | *Category* |
|---|---|---|---|---|
| 700 | 1.524 | 50.8 | G | 1 |
| 700 | 1.524 | 50.8 | B | 2 |
| 700 | 1.524 | 101.6 | G | 1 |
| 700 | 1.524 | 101.6 | B | 2 |
| 700 | 1.524 | 203.2 | G | 1 |
| 700 | 1.524 | 203.2 | B | 2 |
| 700 | 1.524 | 254 | G | 1 |
| 700 | 1.524 | 254 | B | 2 |

The ART gained first experience on the simulation data and later, the neural network inspected the incoming signals and continued to assign new categories when different types of signals were encountered. After simulation training, the neural network started to monitor the experimental data collected at different conditions. The studies focused on selection of the best vigilance, which requires a minimum number of nodes and has acceptable error rate. When the vigilance of 0.98 is used, the network classified the perfect too input data into seven different categories and classified the broken tool input data into four different categories.