Multilayer perceptron network (MLPN).
Globally recurrent networks.

Multilayer perceptron network (MLPN)

A multilayer perceptron network (MLPN) model consists of a feed-forward, layered network of McCulloch and Pitts' neurons. Each neuron in an MLPN has a nonlinear activation function that is often continuously differentiable. Some of the most frequently used activation functions for MLPN include sigmoid function, for instance the hyperbolic tangent function.

A typical MLPN configuration is depicted in Fig. 4.1. Each circle represents an individual neuron. These neurons are organized in layers, labeled as the hidden layer #1, hidden layer #2 and the output layer in this figure. While the inputs at the bottom are also labeled as the input layer, there is usually no neuron model implemented in that layer. The name hidden layer refers to the fact that the output of these neurons will be fed into upper layer neurons and, therefore, is hidden from the user who only observes the output of neurons at the output layer. Fig. 4.1 illustrates a popular configuration of MLPN where interconnections are provided only between neurons of successive layers in the network. In practice, any acyclic interconnections between neurons are allowed.

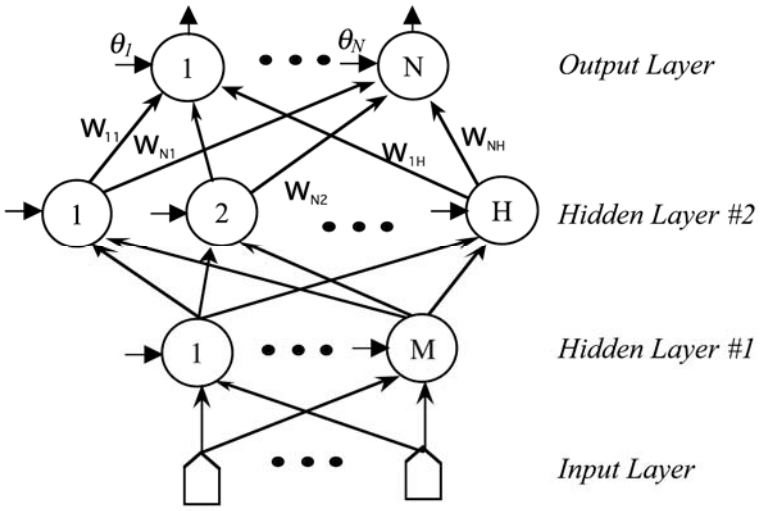


Fig. 4.1. A three-layer multilayer perceptron network configuration

The input signal propagates through the network in a forward direction, on a layer-by-layer basis.

Multilayer perceptron networks have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the error back-propagation algorithm. This algorithm is based on the error-correction learning rule. As such, it may be viewed as a generalization of an equally popular adaptive filtering algorithm: the ubiquitous least-mean-square (LMS) algorithm.

A multilayer perceptron network has three distinctive characteristics:

1. The model of each neuron in the network includes a ***nonlinear activation function.*** The important point to emphasize here is that the nonlinearity is smooth (i.e., differentiable everywhere), as opposed to the hard-limiting used in Rosenblatt's perceptron. A commonly used form of nonlinearity, that satisfies this requirement, is a sigmoidal nonlinearity.

2. The network contains one or more layers of ***hidden neurons*** that are not part of the input or output of the network. These hidden neurons enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns (vectors).

3. The network exhibits high degrees of ***connectivity*****, determined by the synapses of the network**. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

Universal Approximation Theorem

A multilayer perceptron trained with the back-propagation algorithm may be viewed as a practical vehicle for performing a nonlinear input-output mapping of a general nature.

To be specific, let m_0, denote the number of input (source) nodes of a multilayer perceptron, and let $M = m_L$ denote the number of neurons in the output layer of the network. The input-output relationship of the network defines a mapping from an m_0-dimensional Euclidean input space to an M-dimensional Euclidean output space, which is infinitely continuously differentiable when the activation function is likewise. In assessing the capability of the multilayer perceptron from this viewpoint of input-output mapping, the following fundamental question arises:

What is the minimum number of hidden layers in a multilayer perceptron with an input-output mapping that provides an approximate realization of any continuous mapping?

The answer to this question is embodied in *the universal approximation theorem* for a nonlinear input-output mapping, which may be stated as:

Let $\varphi(\bullet)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$. Then, given any function $f \in C(I_{m_0})$ and $\varepsilon > 0$ there exist an integer M and sets of real constants α_i, b_i and ω_{ij}, where $i = 1, ..., m_1$, $j = 1, ..., m_0$ such that we may define

$$F(x_1, ..., x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right) \tag{4.1}$$

as an approximate realization of the function $f(\bullet)$; that is,

$$\left| F(x_1, ..., x_{m_0}) - f(x_1, ..., x_{m_0}) \right| < \varepsilon$$

for all $x_1, x_2, \ldots, x_{m_0}$ that lie in the input space.

The universal approximation theorem is directly applicable to multilayer perceptron networks.

We first note that the logistic function $1/[1 + \exp(-v)]$ used as the nonlinearity in a neuronal model for the construction of a multilayer perceptron is indeed a nonconstant, bounded, and monotone-increasing function; it therefore satisfies the conditions imposed on the function $\varphi(\bullet)$.

Next, we note that Eq. (4.1) represents the output of a multilayer perceptron described as follows:

- The network has m_0 input nodes and a single hidden layer consisting of m_1 neurons; the inputs are denoted by $x_1, x_2, \ldots, x_{m_0}$.
- Hidden neuron i has synaptic weights $\omega_{i1}, ..., \omega_{m_0}$ and bias b_i.
- The network output is a linear combination of the outputs of the hidden neurons, with $\alpha_1, \ldots, \alpha_{m_0}$ defining the synaptic weights of the output layer.

The universal approximation theorem is an existence theorem in the sense that it provides the mathematical justification for the approximation of an arbitrary continuous function as opposed to exact representation. Equation (4.1), which is the backbone of the theorem, merely generalizes approximations by finite Fourier series. **In effect, the theorem states that a single hidden layer is sufficient for a multilayer perceptron to compute a uniform ε approximation to a given**

training set represented by the set of inputs x_1, x_2, …, x_{m_0} and a desired (target) output $f(x_1, x_2, ..., x_{m_0})$.

However, the theorem does not say that a single hidden layer is optimum in the sense of learning time, ease of implementation or (more importantly) generalization.

Practical Considerations of MLPN

The universal approximation theorem is important from a theoretical viewpoint, because it provides the necessary mathematical tool for the viability of feedforward networks with a single hidden layer as a class of approximate solutions. Without such a theorem, we could conceivably be searching for a solution that cannot exist. **However, the theorem is not constructive, that is, it does not actually specify how to determine a multilayer perceptron with the stated approximation properties.**

The universal approximation theorem assumes that the continuous function to be approximated is given and that a hidden layer of unlimited size is available for the approximation. Both of these assumptions are violated in most practical applications of multilayer perceptrons.

The problem with MLPN using a single hidden layer is that the neurons therein tend to interact with each other globally. In complex situations this interaction makes it difficult to improve the approximation at one point without worsening it at some other point.

On the other hand, **with two hidden layers the approximation (curve-fitting) process becomes more manageable**. In particular, we may proceed as follows:

- **Local features are extracted in the first hidden layer.** Specifically, some neurons in the first hidden layer are used to partition the input space into regions, and other neurons in that layer learn the local features characterizing those regions.
- **Global features are extracted in the second hidden layer.** Specifically, a neuron in the second hidden layer combines the outputs of neurons in the first hidden layer operating on a particular region of the input space, and thereby learns the global features for that region and outputs zero elsewhere.

Recurrent neural networks

There are two basic methods of incorporating feedbacks to a neural network:

– global feedback encompassing the whole network,

– local feedback at the level of a single neuron inside the network.

Neural networks with one or more feedbacks are referred to as recurrent networks. As a result of feedbacks introduced to the network structure, it is possible to accumulate the information and use it later.

Feedbacks can be either of a global or a local type. **Taking into account the possible location of feedbacks, recurrent networks can be divided as follows**:

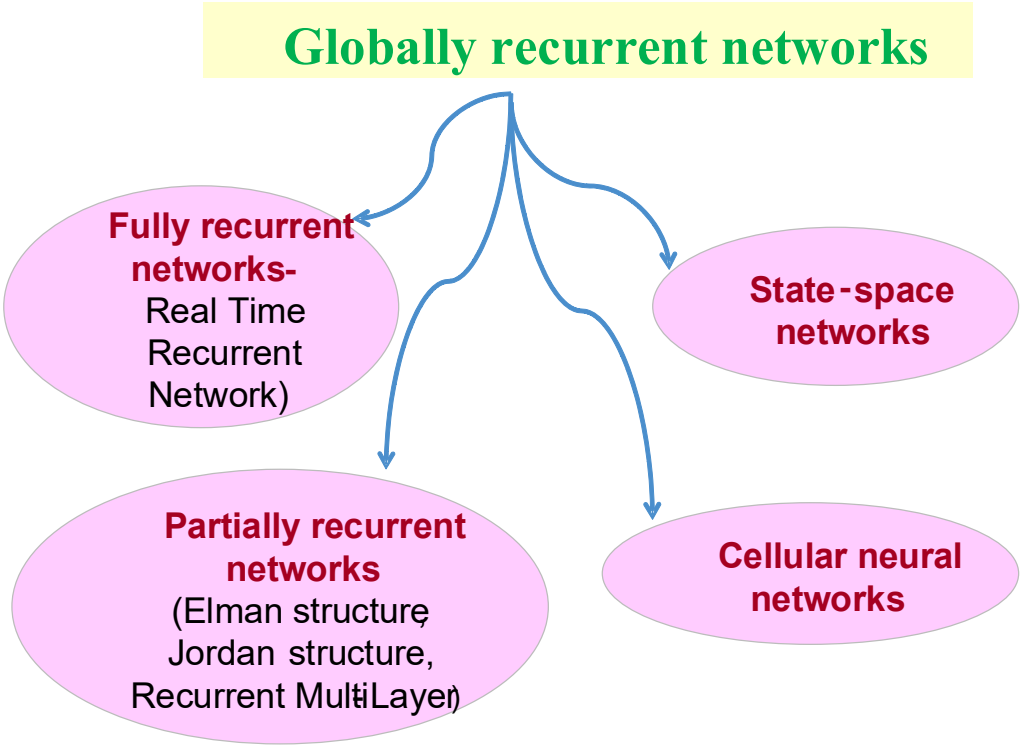
• Globally recurrent networks. There are feedbacks allowed between neurons of different layers or between neurons of the same layer. Such networks incorporate a static multi-layer perceptron or parts of it. Moreover, they exploit the non-linear mapping capability of the multi-layer perceptron. Basically, four kinds of networks can be distinguished:

– fully recurrent networks (the Real Time Recurrent Network (RTRN));

– partially recurrent networks (the Elman structure, the Jordan structure, Recurrent Multi-Layer Perceptron (RMLP));

– state-space networks;

– cellular neural networks (CNN).

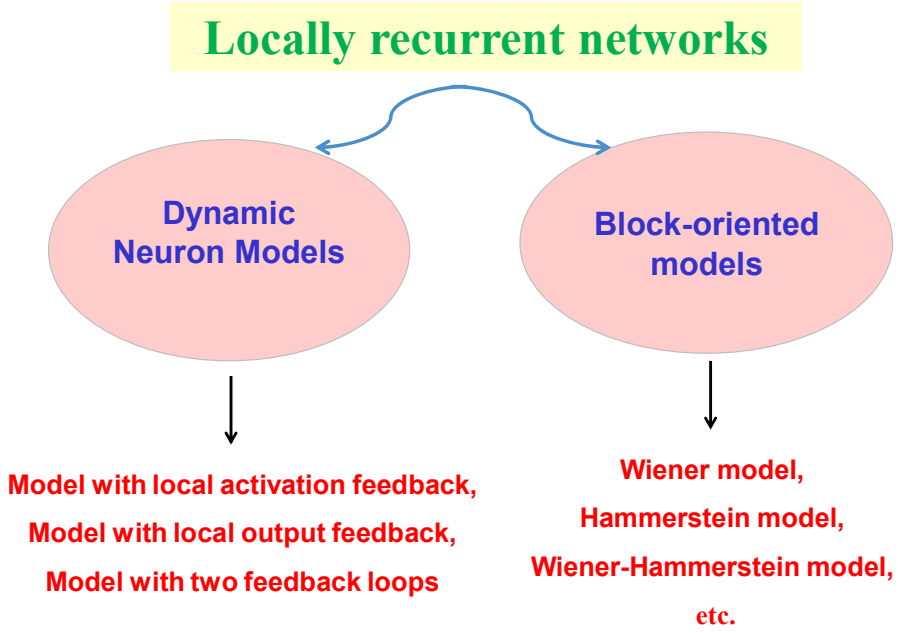


• **Locally recurrent networks**. **There are feedbacks only inside neuron models.** This means that there are neither feedback connections between neurons of successive layers nor lateral links between neurons of the same layer.

One can be distinguished two kinds of locally recurrent networks, namely,

– networks have a structure similar to static feedforward ones, but consist of the so-called dynamic neuron models;

– block oriented stochastic models consisting of static nonlinear and dynamics linear modules (the Hammerstein structure, the Wiener structure, the Hammerstein-Wiener structure and so on).



Globally recurrent networks. Fully Recurrent Networks.

The most general architecture of the recurrent neural network was proposed by Williams and Zipser. This structure is often called **the Real Time Recurrent Network (RTRN)**, because it has been designed for real time signal processing.

Any connections between neurons are allowed. Thus, a fully connected neural architecture is obtained.

A fully connected recurrent neural network, known as the Williams-Zipser network is shown in Fig. 4.2.

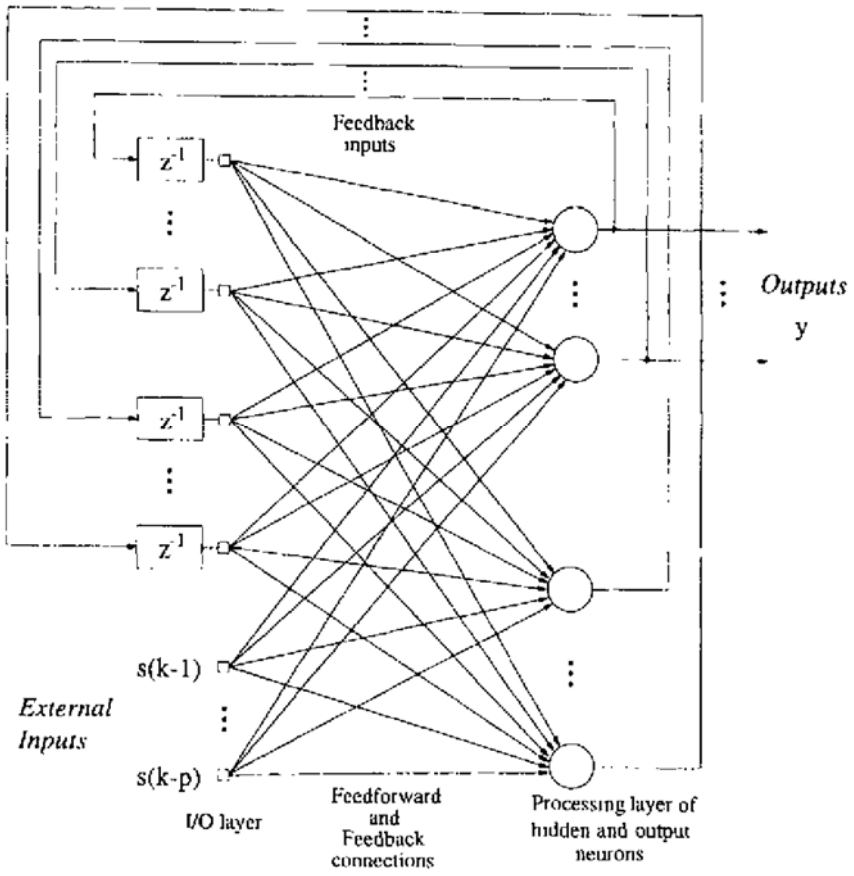


Fig. 4.2. A fully connected recurrent neural network

We give a detailed introduction to this architecture. This network consists of three layers: the input layer, the processing layer and the output layer. For each neuron $i, i = 1, 2, ..., N$, the elements u_j, $j = 1, 2, ..., p + N + 1$, of the input vector to a neuron u, are weighted, then summed to produce an internal activation function of a neuron v, which is finally fed through a nonlinear activation function F, to form the output of the *i*-th neuron y_i. The function F is a monotonically increasing sigmoid function with slope β, as for instance the logistic function,

$$F(v) = \frac{1}{1 - \exp(-\beta v)}.$$

At the time instant *k*, for the *i*-th neuron, its weights form weight vector $\mathbf{W}_i^T(k) = \left[w_{i,1}(k), ..., w_{i,p+N+1}(k) \right]$, where *p* is the number of external inputs, *N* is the number of feedback connections and $(\bullet)^T$ denotes the vector transpose operation. One additional element of the weight vector $\mathbf{W}_i^T(k)$ is the bias input weight. The feedback

consists of the delayed output signals of the RTRN. The following equations fully describe the RTRN from Fig. 4.2,

$$y_i(k) = F\left(v_i(k)\right),\ i = 1, 2, ..., N,$$

$$v_i(k) = \sum_{l=1}^{p+N+1} w_{i,l}(k) u_l(k),$$

$$\mathbf{U}_i^T(k) = [s(k-1), ..., s(k-p), 1, y_1(k-1),$$

$$y_2(k-1), ..., y_N(k-1)],$$

where the vector U comprises both the external and feedback inputs to a neuron, as well as the unity valued constant bias input.

The fundamental **advantage** of such networks is **the possibility of approximating the wide class of dynamic relations**.

Such a kind of networks, however, exhibits some well-known **disadvantages**. First of them is a **large structural complexity**. Also, **the training of the network is usually complex and slowly convergent**. Moreover, **there are problems with keeping network stability**. Generally, this dynamic structure seems to be too complex for practical applications. Moreover, the fixed relation between the number of states and the number of neurons does not allow one to adjust the dynamics order and non-linear properties of the model separately. Bearing in mind these disadvantages, fully recurrent networks are rather not used in engineering practice of non-linear system identification.

Globally recurrent networks. Partially recurrent networks.

Partially recurrent networks have a less general character. Contrary to the fully recurrent network, **the architecture of partially recurrent networks is based on a feedforward multi-layer perceptron consisting of an additional layer of units called the context layer. Neurons of this layer serve as internal states of the model.**

Among many proposed structures, three partially recurrent networks have received considerable attention:

- **the Elman structure,**
- **the Jordan structures,**
- **the Recurrent Multi-Layer Perceptron (RMLP).**

Partially recurrent networks possess over fully recurrent networks the advantage that their recurrent links are more structured, which leads to **faster training and fewer stability problems. Nevertheless, the number of states is still strongly related to the number of hidden (for Elman) or output (for Jordan) neurons, which severely restricts their flexibility.**

The Elman network is probably the best-known example of a partially recurrent neural network. The realization of such networks is considerably less expensive than in the case of a multi-layer perceptron with tapped delay lines.

The scheme of the Elman network is shown in Fig. 4.3.

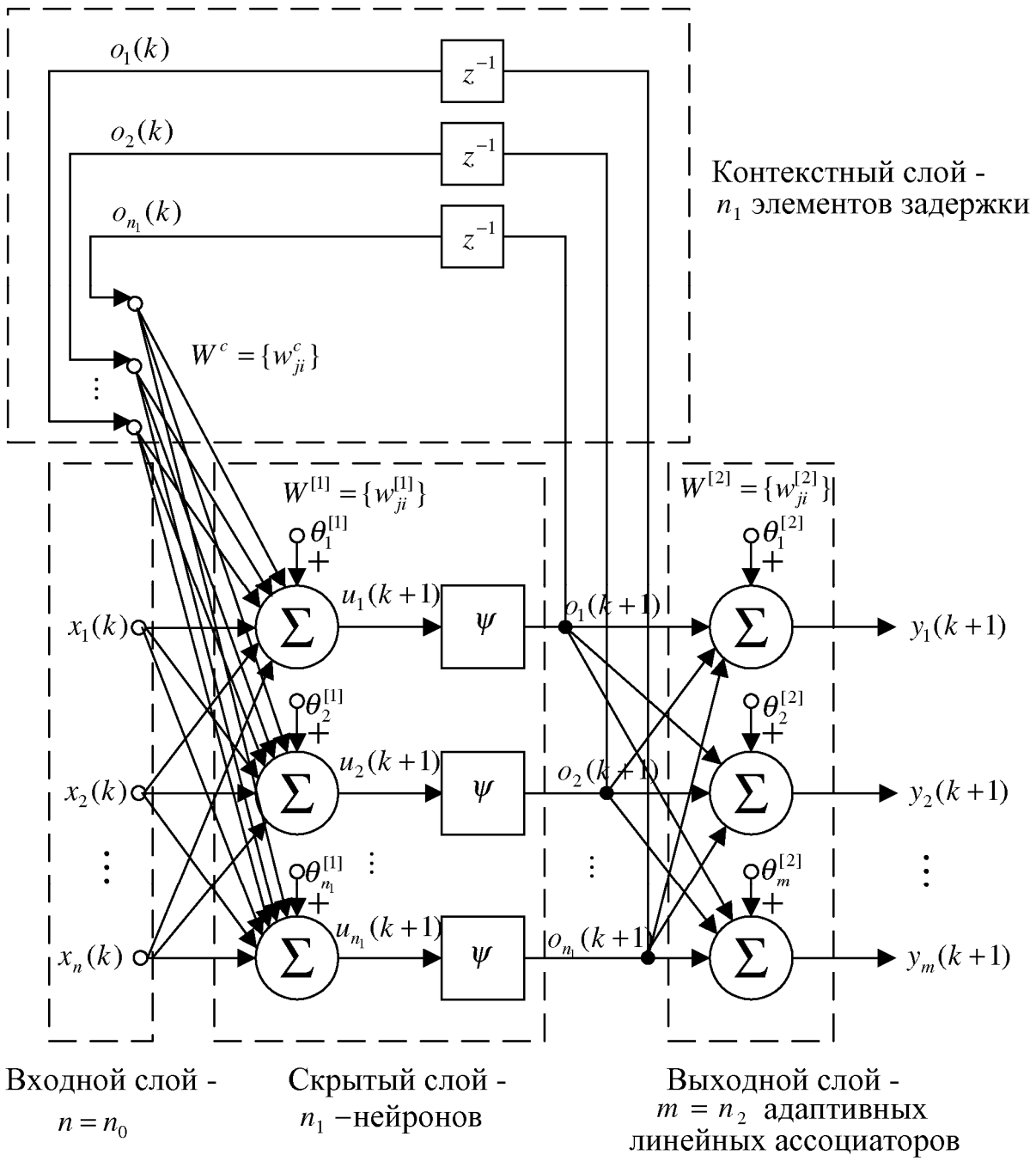


Fig. 4.3. The Elman recurrent network

The Elman network is described by a system of recurrent equations:

$$\begin{cases} u_j(k+1) = \sum_{i=1}^{n} w_{ji}^{[1]} x_i(k) + \sum_{i=1}^{n_1} w_{ji}^{c} o_i(k) + \theta_j^{[1]}, \\ o_j(k+1) = \psi(u_j(k+1)) = \tanh u_j(k+1), \quad j = 1,2,\ldots,n_1, \end{cases}$$

$$y_j(k+1) = \sum_{i=1}^{n_1} w_{ji}^{[2]} o_i(k+1) + \theta_j^{[2]}, \quad j = 1,2,\ldots,m$$

or in matrix form

$$o(k+1) = \Psi(W^{[1]} x(k) + W^{c} o(k) + \theta^{[1]}),$$

$$y(k+1) = W^{[2]} o(k) + \theta^{[2]},$$

where W^C is the matrix with size $(n_1 \times n_2)$ of the context layer synaptic weights.

The Elman network spreads to control systems of moving objects, to detecting a change in the signal characteristics.

The Jordan network is presented in Fig. 4.4. In this case, feedback connections from the output neurons are fed to the context units. It consists of a multilayer perceptron with one hidden layer and a feedback loop from the output layer to an additional input called the context layer. In the context layer, there are self-recurrent loops with the forgetting coefficient α, $0 \le \alpha \le 1$.

Thus, the signal at the output of context neurons is determined not only the current output signals of the network as well as weighted previous output signals. This feature of the Jordan network provides more "deep" memory of this network compared to the Elman network.

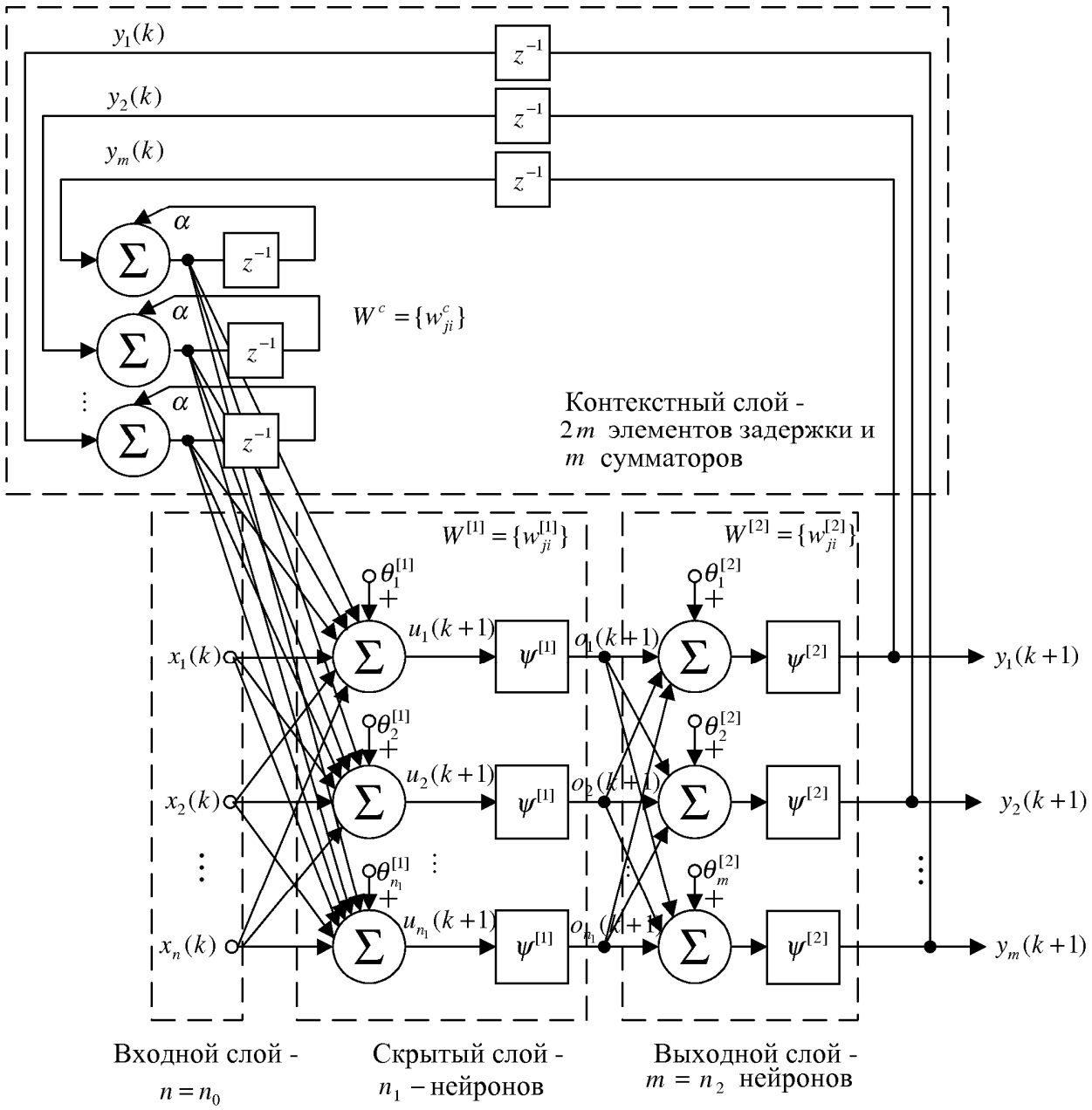


Fig. 4.4. The Jordan network

The Jordan network is described by a system of recurrent equations:

$$\begin{cases} u_j(k+1) = \sum_{i=1}^{n} w_{ji}^{[1]} x_i(k) + \sum_{i=1}^{m} w_{ji}^{c} (y_i(k) + \alpha y_i(k-1)) + \theta_j^{[1]}, \\ o_j(k+1) = \psi^{[1]}(u_j(k+1)), \quad j = 1,2,\ldots,n_1, \end{cases}$$

$$y_j(k+1) = \psi^{[2]}\left(\sum_{i=1}^{n_1} w_{ji}^{[2]} o_i(k+1) + \theta_j^{[2]} \right), \quad j = 1,2,\ldots,m$$

or in matrix form

$$o(k+1) = \Psi^{[1]}(W^{[1]} x(k) + W^c (y(k) + \alpha y(k-1)) + \theta^{[1]}),$$

$$y(k+1) = \Psi^{[2]}\left(W^{[2]} o(k) + \theta^{[2]}\right) =$$

$$= \Psi^{[2]}\left(W^{[2]}(\Psi^{[1]}(W^{[1]} x(k) + W^{c}(y(k) + \alpha y(k-1)) + \theta^{[1]}) + \theta^{[2]})\right)$$

This network solves the same task class like the Elman network, however, it has the best approximating and predictive properties due to the deeper memory and an additional layer of non-linear activation functions.

The Jordan network has been successfully applied to recognize and differentiate various output time-sequences or to classify various sequences.

Another architecture can be found in the recurrent network elaborated by Parlos (Fig. 4.5). A **Recurrent Multi-Layer Perceptron (RMLP**) is designed based on the multi-layer perceptron network, and by adding delayed links between neighboring units of the same hidden layer (recurrent links), including unit feedback on itself (cross-talk links).

The RMLP network is presented in Fig. 4.5.

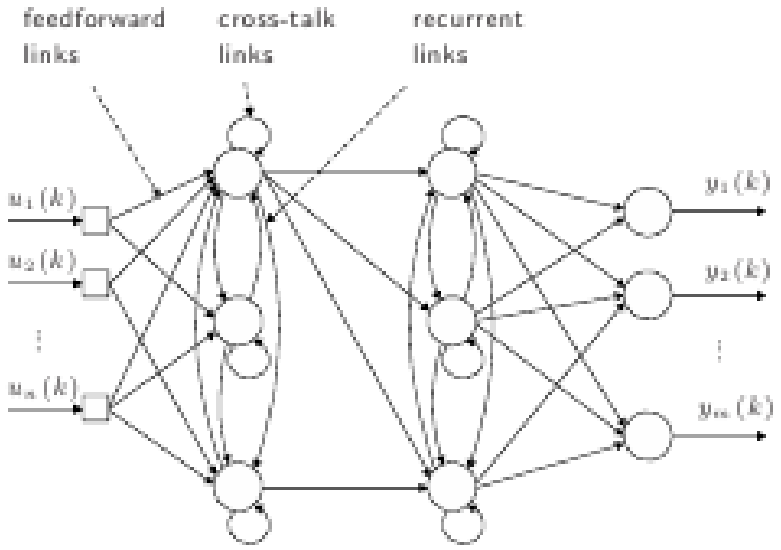


Fig. 4.5. Recurrent Multi-Layer Perceptron

Empirical evidence indicates that by using delayed recurrent and cross-talk weights **the RMLP network is able to emulate a large class of non-linear dynamic systems**. The feedforward part of the network still maintains the well-known curve-fitting properties of the multi-layer perceptron, while the feedback part provides its dynamic character. Moreover, the usage of the past process observations is not necessary, because their effect is captured by internal network states.

The RMLP network has been successfully used as a model for dynamic system identification. However, **a drawback of this dynamic structure is increased network complexity strictly dependent on the number of hidden neurons and the resulting long training time.**

Globally recurrent networks. State-space networks.

Fig 4.6 shows another type of recurrent neural networks known as the state-space neural network.

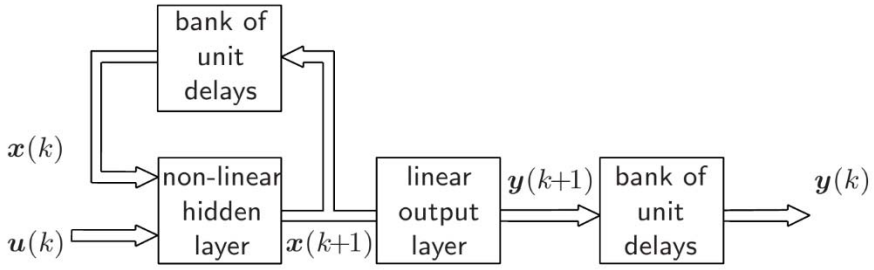


Fig. 4.6. Block scheme of the state-space neural network with one hidden layer

The output of the hidden layer is fed back to the input layer through a bank of unit delays. The number of unit delays, used here, determines the order of the system. The user can choose how many neurons are used to produce feedback. Let $\mathbf{U}(k) \in R^n$ be the input vector, $\mathbf{X}(k) \in R^q$ – the output of the hidden layer at time *k*, and $\mathbf{Y}(k) \in R^m$ is the output vector.

Then the state-space representation of the neural model presented in Fig. 4.6 is described by equations

$$\mathbf{X}(k+1) = \mathbf{F}\big(\mathbf{X}(k), \mathbf{U}(k)\big),$$

$$\mathbf{Y}(k) = \mathbf{C}\mathbf{X}(k),$$

where $\mathbf{F}$ is the non-linear function characterizing a hidden layer, and $\mathbf{C}$ is the matrix of synaptic weights between hidden and output neurons.

State-space models possess a number of advantages, contrary to fully and partially recurrent networks:

- **State-space models can describe a wide class of non-linear dynamic systems**.
- **The number of states (model order) can be selected independently from the number of hidden neurons.** In this way only those neurons that feed their outputs back

to the input layer through delays are responsible for defining the state of the network. As a consequence, the output neurons are excluded from the definition of the state;

• **Since model states feed the input of the network, they are easily accessible from the outside environment.** This property can be useful when state measurements are available at some time instants (e.g. initial conditions).

The state-space model includes several recurrent structures as special cases.

In spite of the fact that state-space neural networks seem to be more promising than fully or partially neural networks, **in practice a lot of difficulties can be encountered**:

• **The approximation is only valid on compact subsets of the state-space and for finite time intervals, thus interesting dynamic characteristics are not reflected.**

• **Wrong initial conditions can worsen the performance**, especially when short data sets are used for training.

• **Training can become unstable**.

• **The model after training can be unstable**.

In particular, these drawbacks appear in the cases when no state measurements and no initial conditions are available.