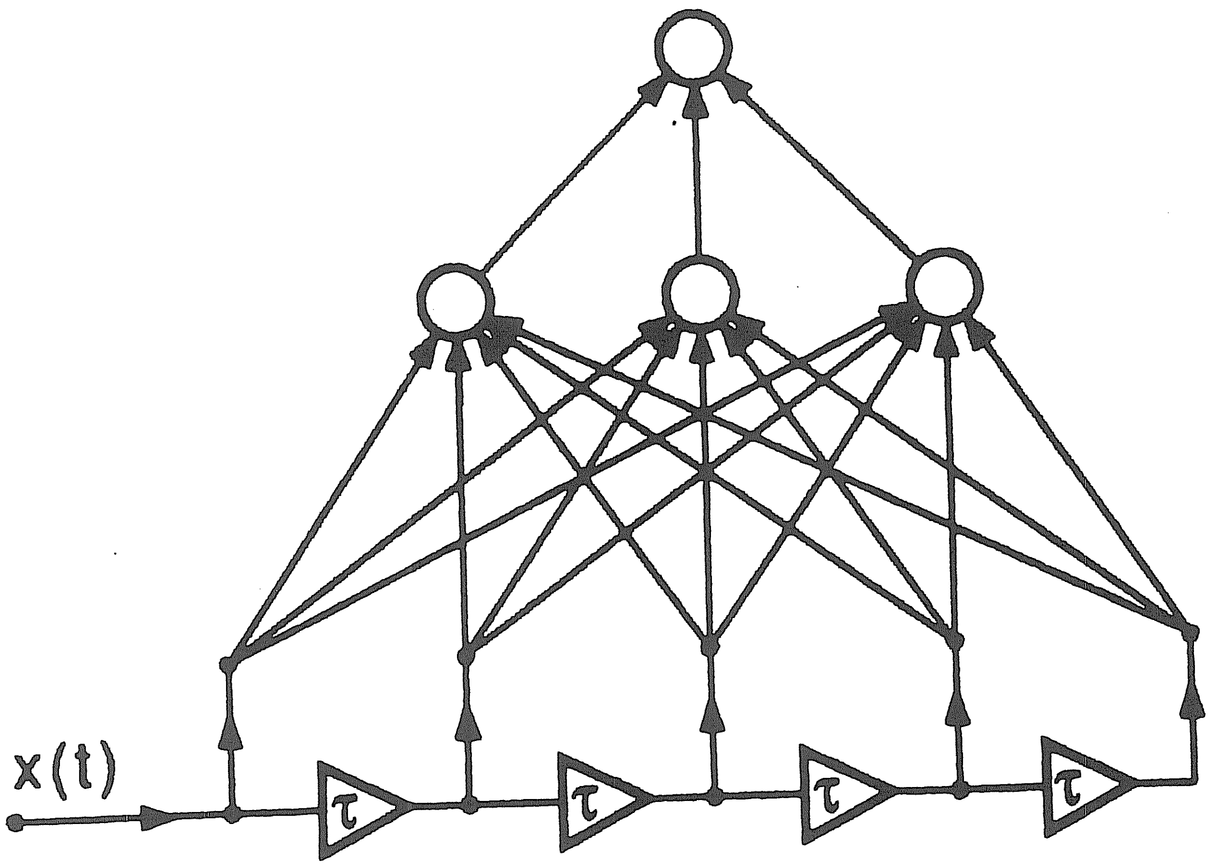


FIGURE 7.3 A time-delay neural network. Only one input $x(t)$ is shown. $x(t)$, $x(t-\tau)$, $\ldots$, $x(t-4\tau)$ are fully connected to a hidden layer.

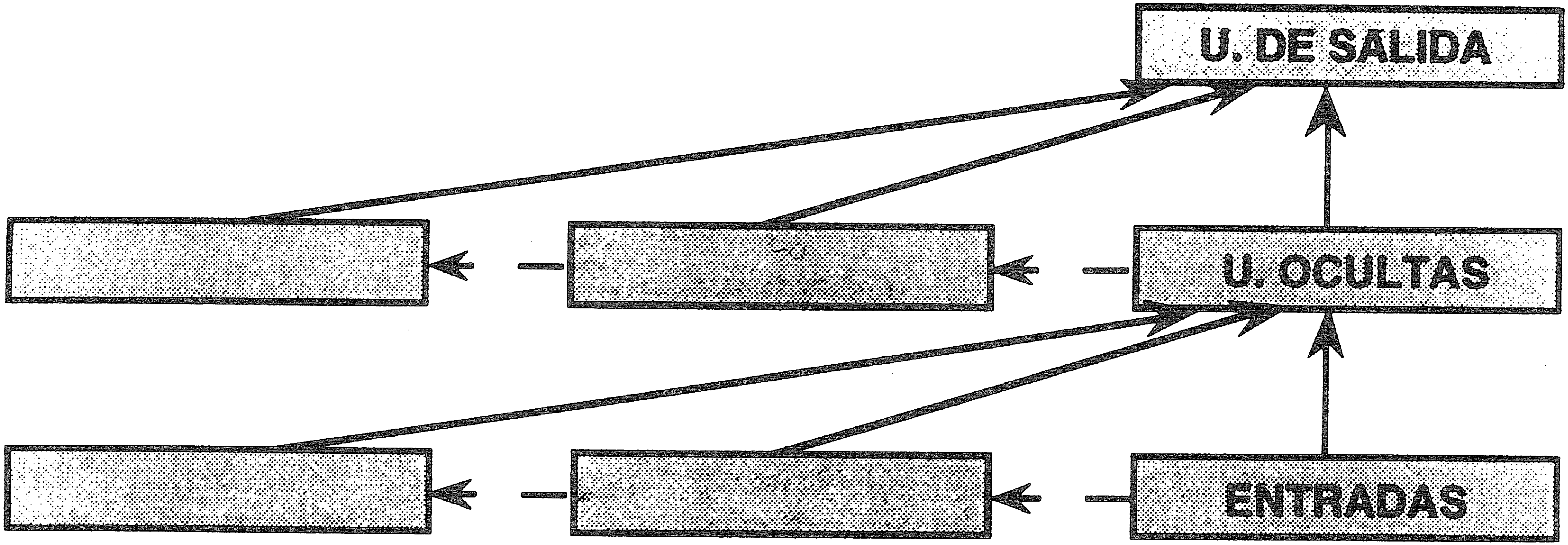


Figura 2.1. Red neuronal con retrasos de tiempo.

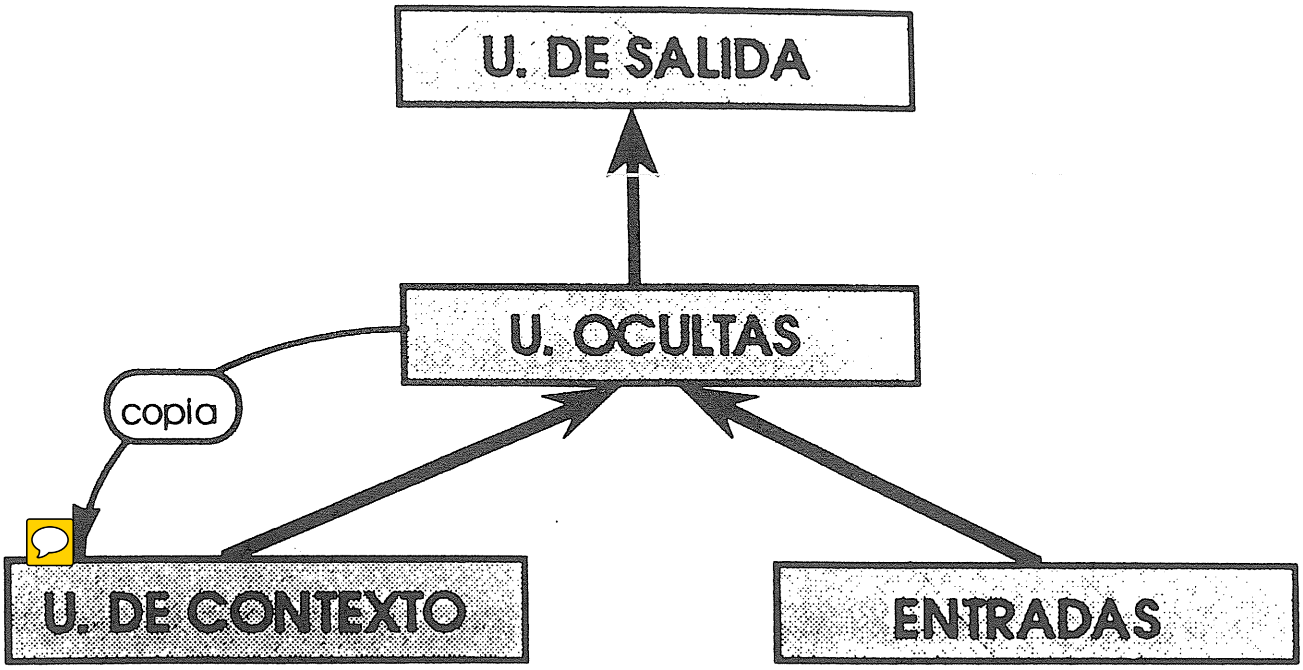


Figura 2.2. Red recurrente simple de Elman.

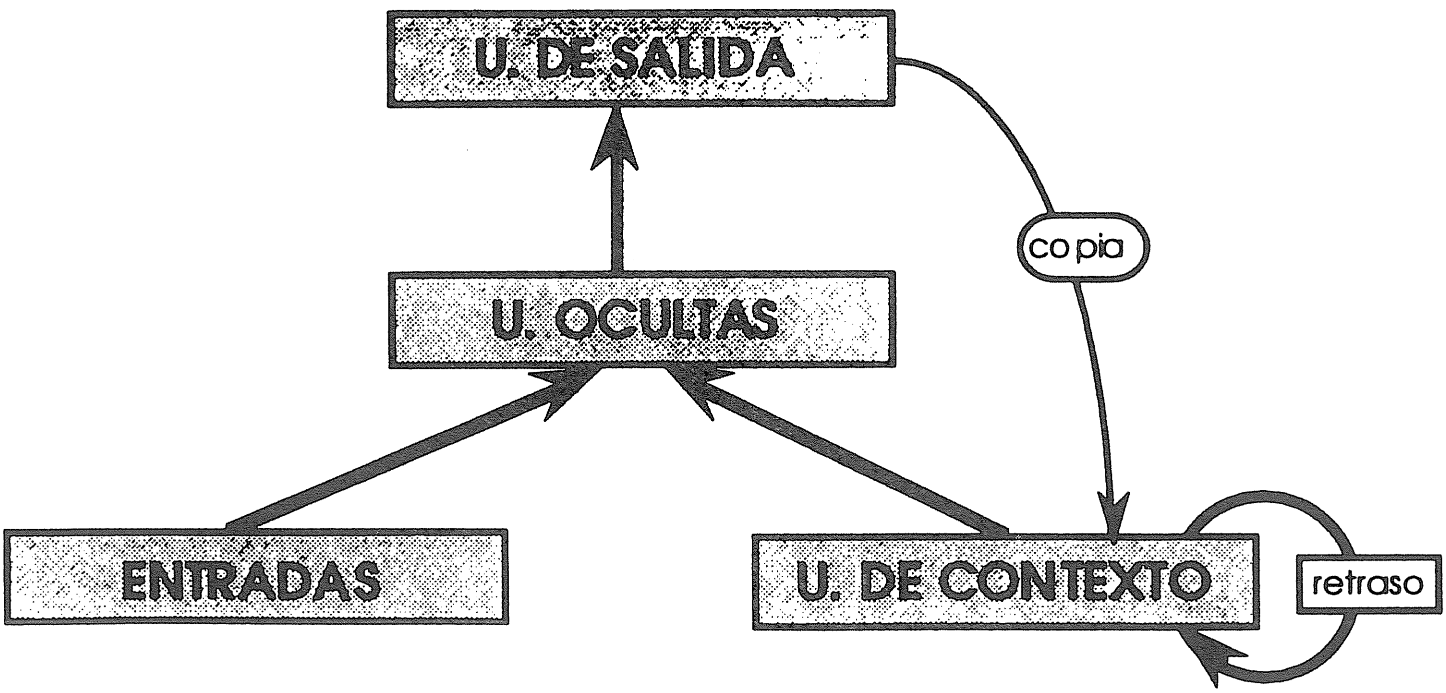


Figura 2.3. Red recurrente simple de Jordan.

$$C_i(t+1) = \alpha C_i(t) + O_i(t) \qquad \alpha < 1$$

$$C_i(t+1) = O_i(t) + \alpha O_i(t-1) + \alpha^2 O_i(t-2) + \ldots = \sum_{t'=0}^{t} \alpha^{(t-t')} O_i(t')$$

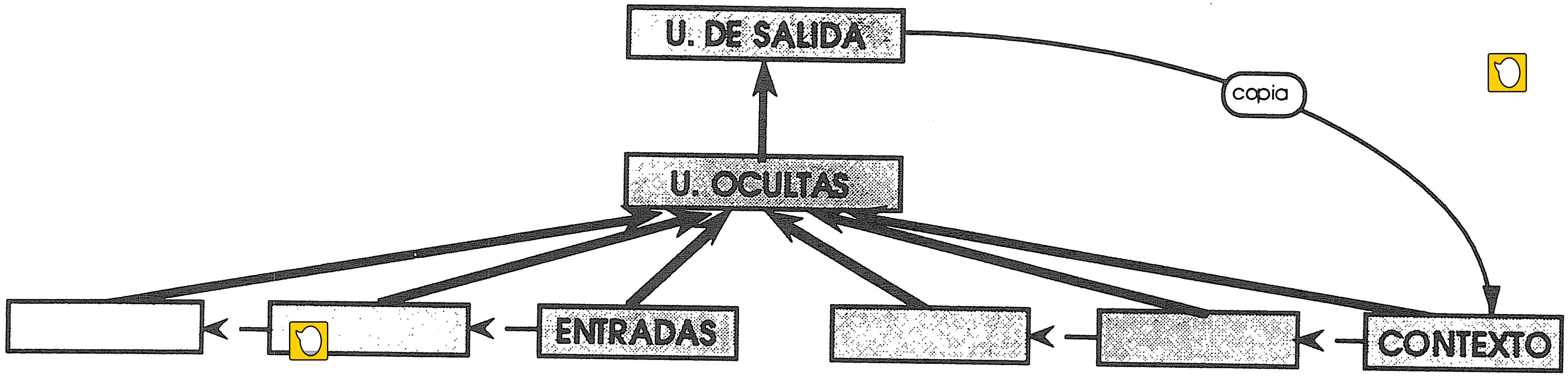


Figura 2.4. Red NARX.

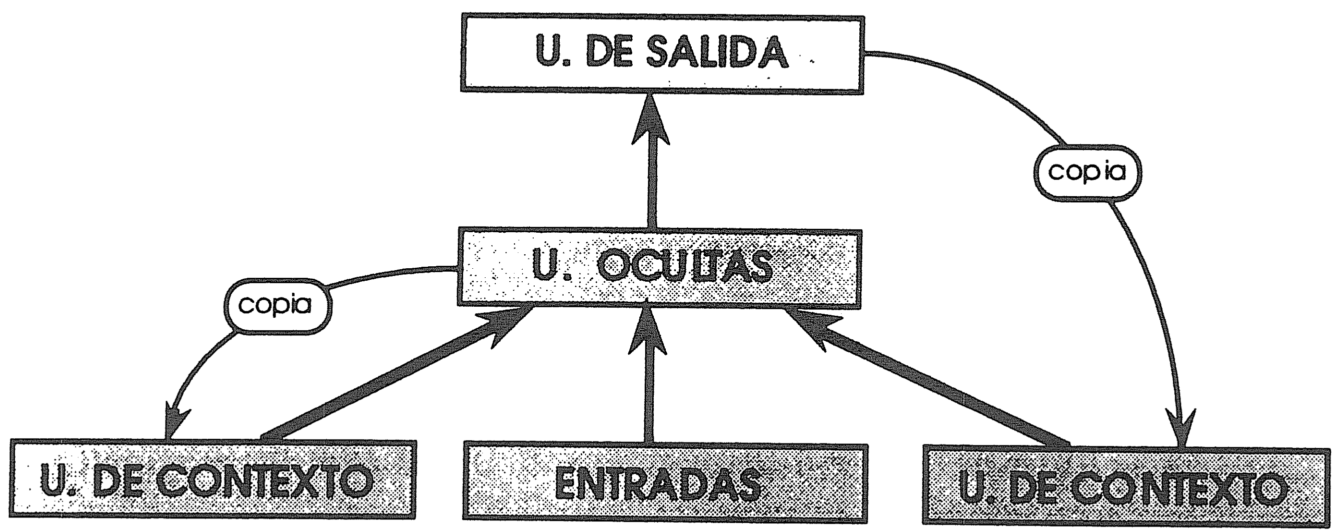


Figura 2.5. Red híbrida Elman-Jordan.

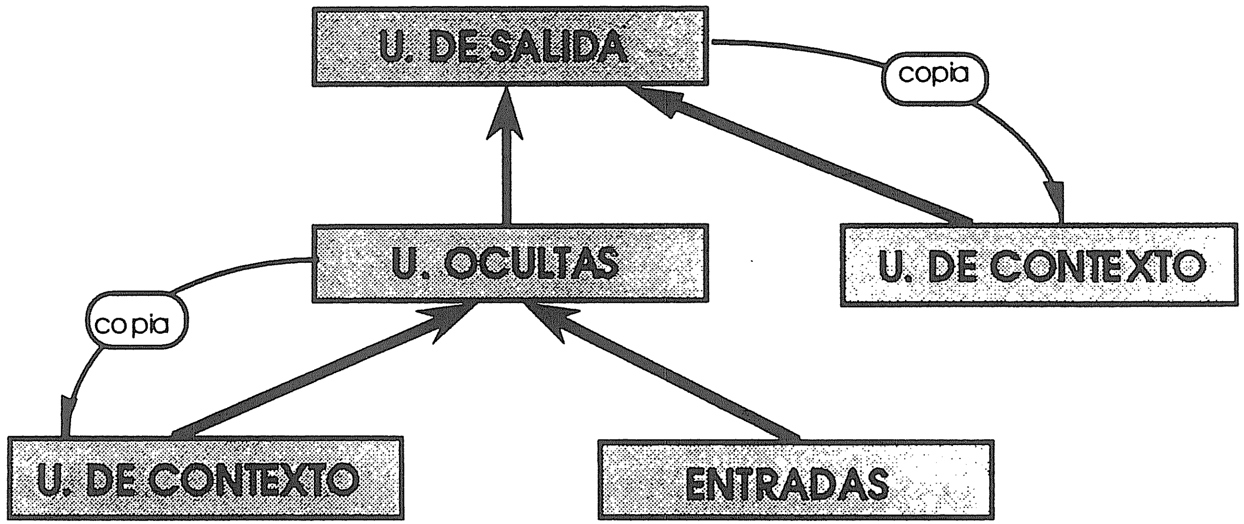


Figura 2.6. Red de Elman Extendida.

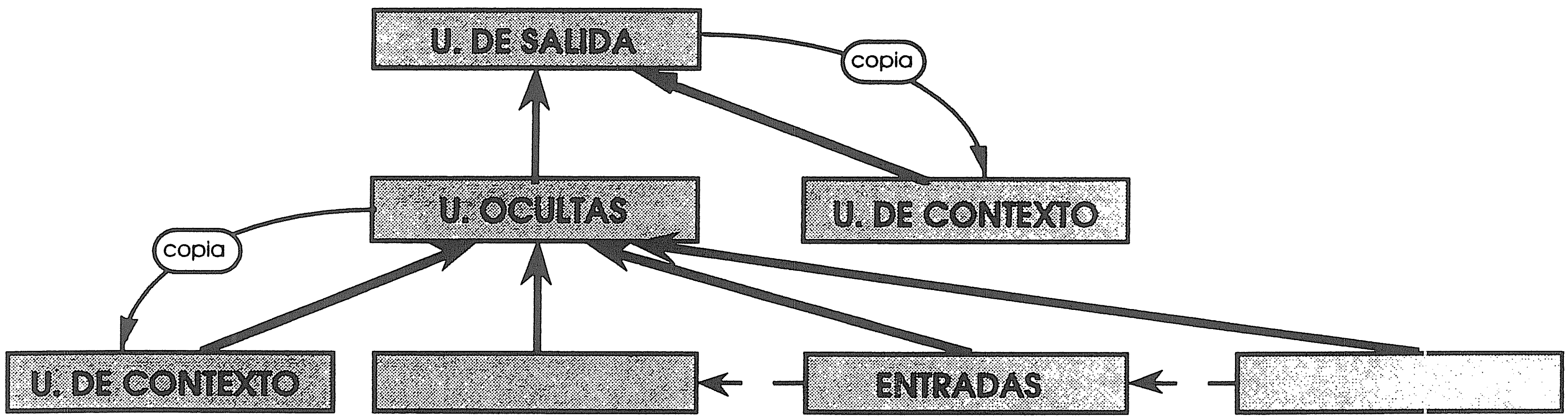
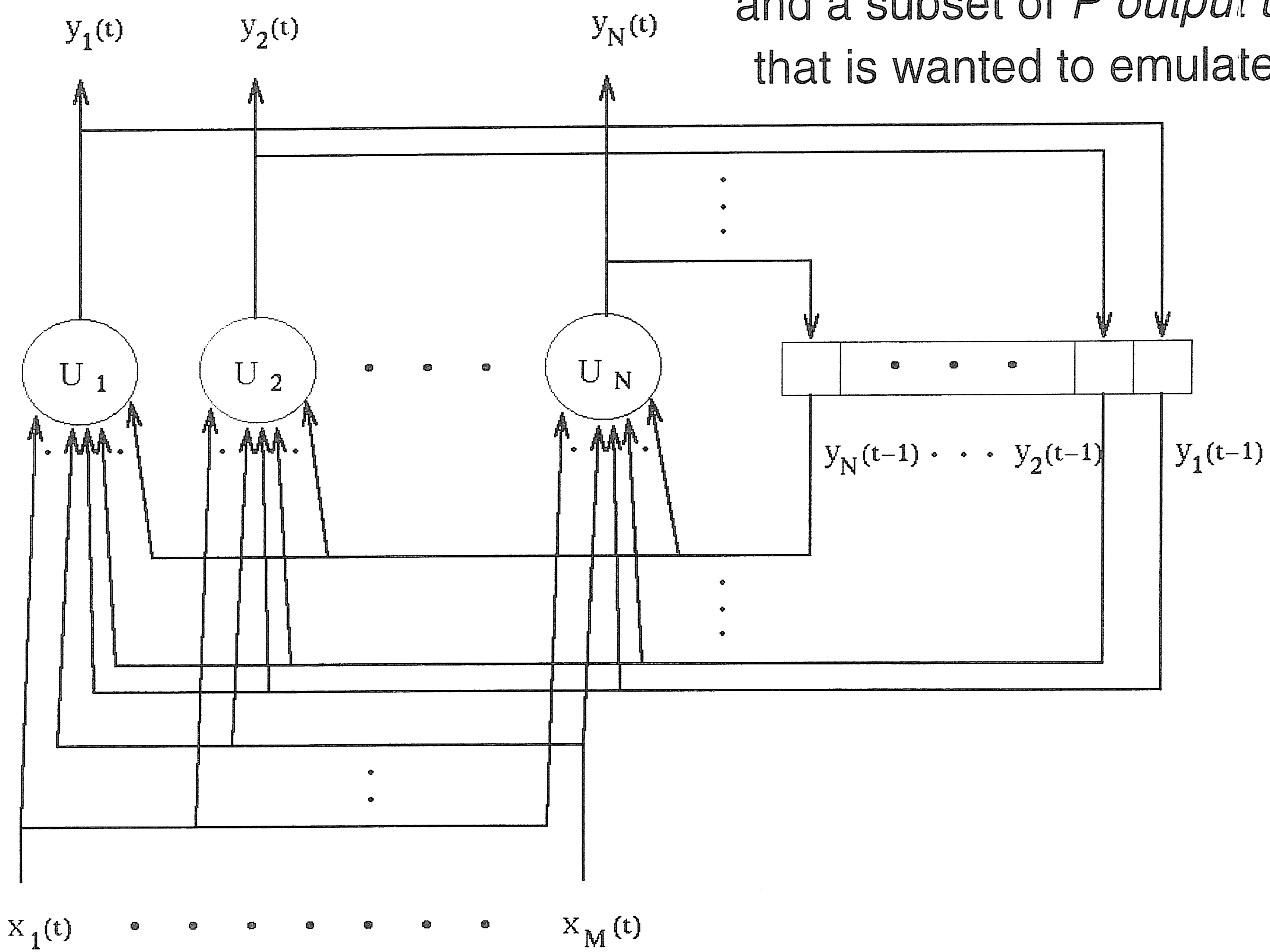


Figura 2.7. Red de Elman Extendida con retrasos de tiempo en la entrada.

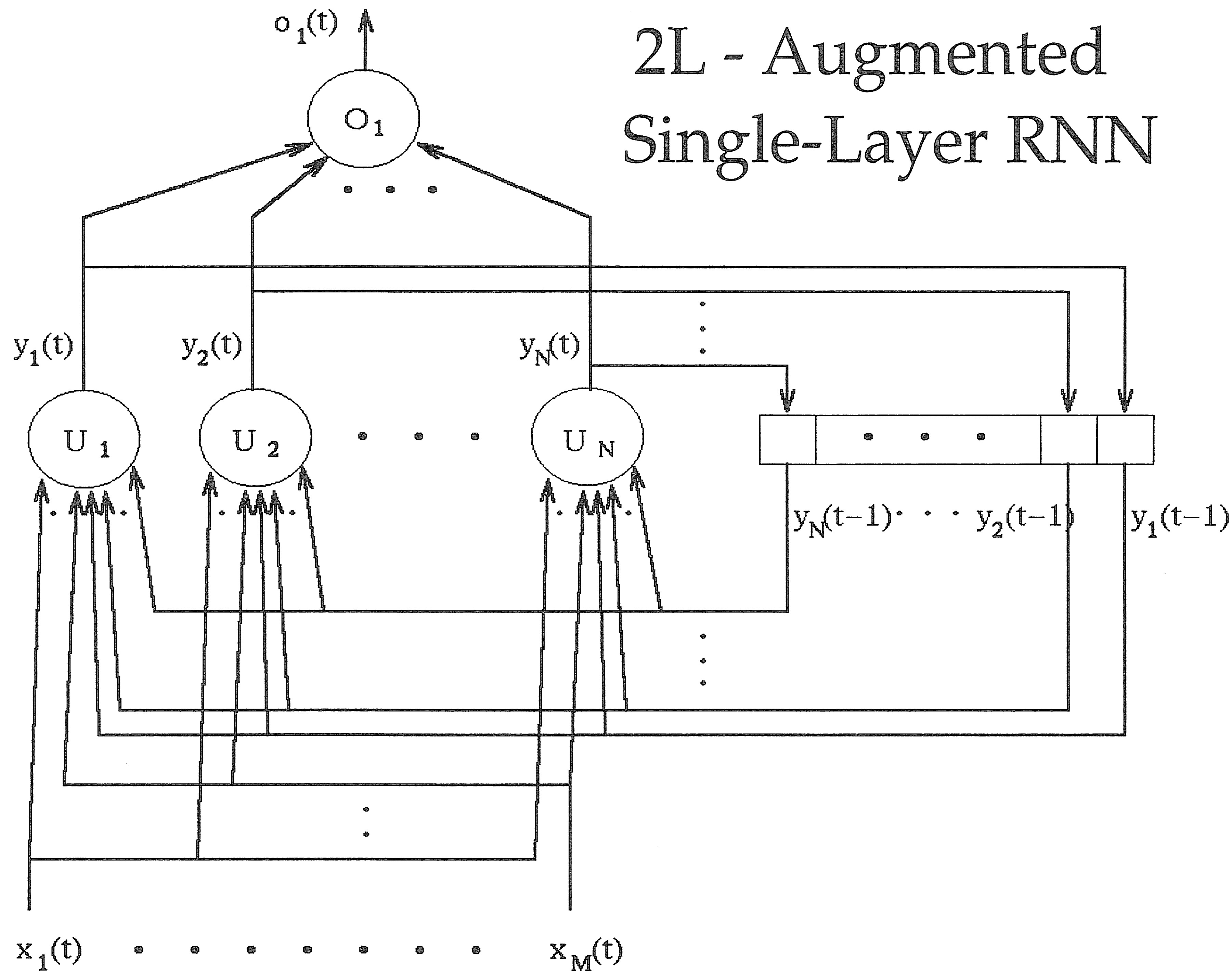
Studied RNN architectures

- *Single-Layer Recurrent NNs* (SLRNNs): a fully-connected recurrent layer of units
  - *First-order* SLRNNs (Williams & Zipser, 89)
  - *Second-order* SLRNNs (Pollack, 91; Giles *et al.*, 92)
- *Augmented Single-Layer Recurrent NNs* (ASLRNNs): an SLRNN of hidden units + one or two feed-forward layers (2L-ASLRNNs, 3L-ASLRNNs)
  - *First-order* ASLRNNs (Elman, 90)
  - *Second-order* ASLRNNs

Let $\mathcal{N}$ be an *SLRNN* with *M input signals*, *N recurrent units*, and a subset of *P output units*, that is wanted to emulate $\mathcal{M}$.



2L - Augmented Single-Layer RNN



First-order vs. second-order 2L-ASLRNNs

⇛ *Recurrent layer*

– *First-order* connections

$$y_k(t) = g_1\left(w_{k0} + \sum_{i=1}^{M} w_{ki} x_i(t) + \sum_{j=1}^{N} w_{k(M+j)} y_j(t-1)\right) \quad \text{for } 1 \le k \le N$$

– *Second-order* connections

$$y_k(t) = g_1\left(w_{k0} + \sum_{i=1}^{M} \sum_{j=1}^{N} w_{kij} x_i(t) y_j(t-1)\right) \quad \text{for } 1 \le k \le N$$

⇛ *Output layer*

$$o_l(t) = g_2\left(w_{l0}^{o} + \sum_{j=1}^{N} w_{lj}^{o} y_j(t)\right) \quad \text{for } 1 \le l \le P$$

First-order SLRNN

$$y_k(t) = g\left(\sum_{i=1}^{M} w_{ki}\, x_i(t) + \sum_{j=1}^{N} w_{k(M+j)}\, y_j(t-1) \right) \quad \text{for } 1 \le k \le N$$

$$y_k(t) = g\left(\sum_{i=1}^{M+N} w_{ki}\, z_i(t) \right) \quad \text{where} \quad z_i(t) = \begin{cases} x_i(t) & \text{if } i \le M \\ y_{i-M}(t-1) & \text{if } i > M \end{cases}$$

Output units at time t: $T(t) = \{ k \mid 1 \le k \le N,\ \exists\, d_k(t) \}$

Local error:
$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases}$$

Total squared error at time t: $E(t) = \frac{1}{2} \sum_{k=1}^{N} [e_k(t)]^2$

Total squared error over a whole training sequence: $E_{total}(1, t_f) = \sum_{t=1}^{t_f} E(t)$

Weight adjustment by gradient descent:

$$\Delta w_{ij} = -\alpha \frac{\partial E_{total}(1, t_f)}{\partial w_{ij}}$$

True gradient computation:

1) BPTT (Rumelhart, Hinton, Williams, 86)
2) RTRL (Williams & Zipser, 89)
3) Schmidhuber's algorithm (Schmidhuber, 92)

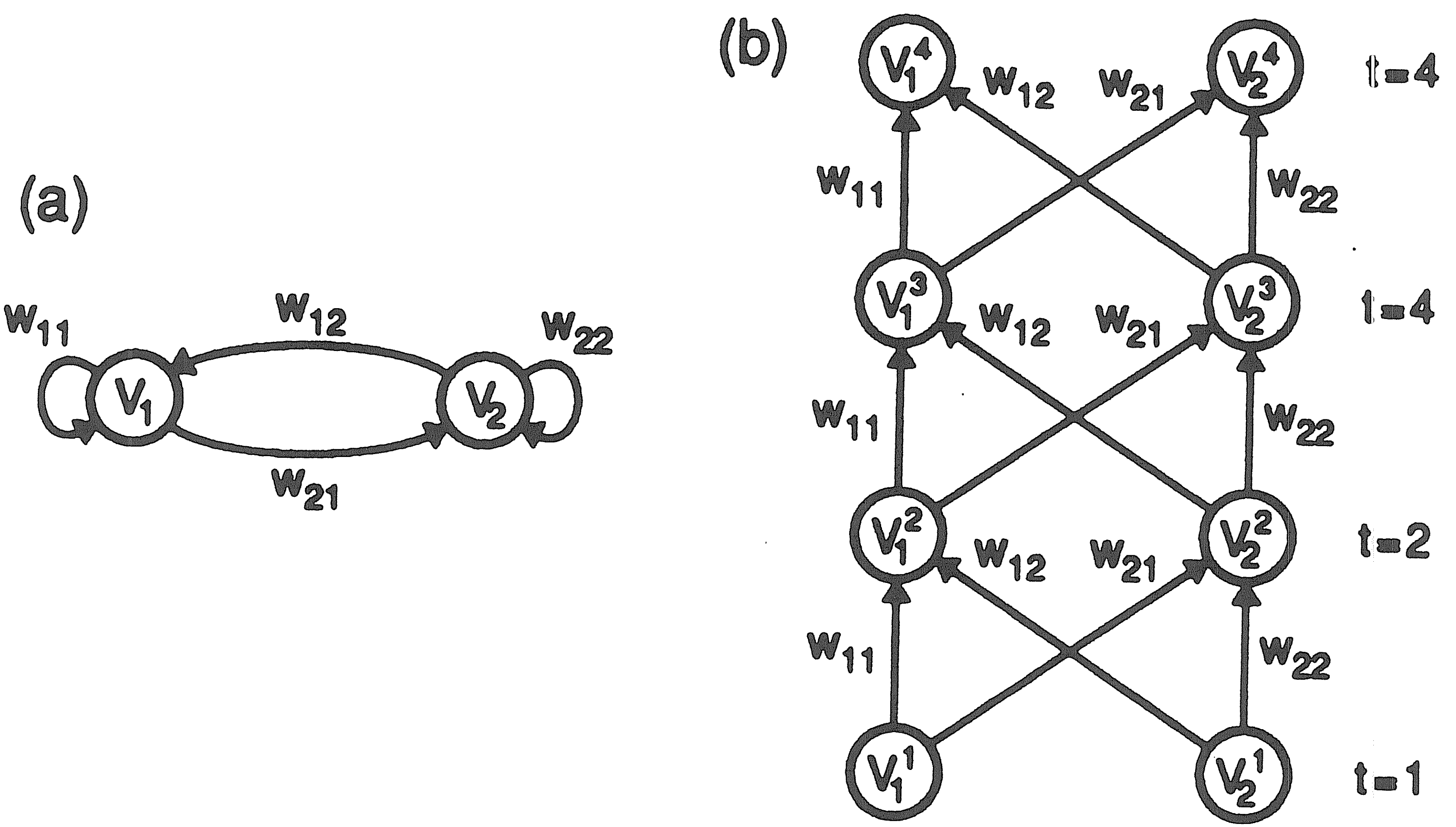


FIGURE 7.6 Back-propagation through time. (a) A recurrent network. (b) A feed-forward network that behaves identically for 4 time steps.

Back-propagation through time (BPTT)

Idea: desplegar la red recurrente en una red multicapa con activación hacia delante con una capa para cada paso de tiempo, pero pesos idénticos entre diferentes capas:

$$w_{ij}(t) = w_{ij} \quad \text{for all} \quad t \in [1, t_f]$$

$$\frac{\partial E_{total}(1,t_f)}{\partial w_{ij}} = \sum_{t=1}^{t_f} \frac{\partial E_{total}(1,t_f)}{\partial w_{ij}(t)} = -\sum_{t=1}^{t_f} \delta_i(t)\, z_j(t)$$

$$\text{where} \quad \delta_i(t) = -\frac{\partial E_{total}(1,t_f)}{\partial \sigma_i(t)}$$

and $\sigma_i(t)$ is the net-input of unit i at time t.

For $1 \leq i \leq N$ and $t_f \geq t \geq 1$:

$$\delta_i(t) = \begin{cases} g'[\sigma_i(t)]\; e_i(t) & \text{if} \quad t = t_f \\ g'[\sigma_i(t)]\left(e_i(t) + \sum_{k=1}^{N} w_{k(M+i)}\, \delta_k(t+1)\right) & \text{if} \quad t < t_f \end{cases}$$

Time complexity: $O(N^2)$ per time step

Space complexity: $O(N^2 + (\underline{\max\{t_f\}} \cdot (N+M)))$!!

Run-Time Recurrent Learning (RTRL)

Permite un ajuste "on-line" de los pesos y secuencias de entrenamiento de longitud arbitraria (no acotada).

$$\Delta w_{ij} = \sum_{t=1}^{t_f} \Delta w_{ij}(t)$$

where $$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha \sum_{k=1}^{N} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}}$$

Let $$p_{ij}^{k}(t) = \frac{\partial y_k(t)}{\partial w_{ij}}$$ for $1 \leq i,k \leq N$, $1 \leq j \leq M+N$.

$$\begin{cases} p_{ij}^{k}(0) = 0 \\ p_{ij}^{k}(t) = g'[\sigma_k(t)] \cdot \left(\delta_{ik}\, z_j(t) + \sum_{l=1}^{N} w_{k(M+l)}\, p_{ij}^{l}(t-1) \right) \text{ for } t>0 \end{cases}$$

Off-line learning : use Δw_{ij} at the end of sequence.

On-line learning : use $\Delta w_{ij}(t)$ at each time step.

Time complexity: $O(N^4)$ per time step !!

Space complexity: $O(N^3)$

Schmidhuber's algorithm

Descompone el cálculo del gradiente en bloques de h pasos de tiempo y combina cálculos "BPTT-like" con cálculos "RTRL-like".

Let $$q_{ij}^{k}(t) = \frac{\partial \sigma_k(t)}{\partial w_{ij}} = \sum_{z=1}^{t} \frac{\partial \sigma_k(t)}{\partial w_{ij}(z)} \quad \text{for } 1 \le i, k \le N;\ 1 \le j \le M+N.$$

$\{q_{ij}^{k}\}$ are updated at each block of h time steps; $q_{ij}^{k}(0) = 0$

$$\frac{\partial E_{total}(1, t_0+h)}{\partial w_{ij}} = \underbrace{\frac{\partial E_{total}(1, t_0)}{\partial w_{ij}}}_{\text{1st term (known)}} + \underbrace{\sum_{t=1}^{t_0} \frac{\partial E_{total}(t_0+1, t_0+h)}{\partial w_{ij}(t)}}_{\text{2nd term}} + \underbrace{\sum_{t=t_0+1}^{t_0+h} \frac{\partial E_{total}(t_0+1, t_0+h)}{\partial w_{ij}(t)}}_{\text{3rd term}}$$

3rd term: $$\sum_{t=t_0+1}^{t_0+h} \frac{\partial E_{total}(t_0+1, t_0+h)}{\partial w_{ij}(t)} = -\sum_{t=t_0+1}^{t_0+h} \delta_i(t)\, z_j(t)$$

where $$\delta_i(t) = -\frac{\partial E_{total}(t_0+1, t_0+h)}{\partial \sigma_i(t)}$$

$$\delta_i(t) = \begin{cases} g'[\sigma_i(t)]\; e_i(t) & \text{if } t = t_0+h \\[2ex] g'[\sigma_i(t)] \left(e_i(t) + \sum_{k=1}^{N} w_{k(m+i)}\, \delta_k(t+1) \right) & \text{if } t_0 \le t < t_0+h \end{cases}$$

2nd term: $\displaystyle\sum_{t=1}^{t_0} \frac{\partial E_{total}(t_0+1, t_0+h)}{\partial w_{ij}(t)} = -\sum_{k=1}^{N} \delta_k(t_0)\, q_{ij}^k(t_0)$

Updating the $\{q_{ij}^k\}$ variables:

$$q_{ij}^k(t_0+h) = \sum_{t=t_0+1}^{t_0+h} \gamma_{ki}(t)\, z_j(t) + \sum_{l=1}^{N} \gamma_{kl}(t_0)\, q_{ij}^l(t_0)$$

$$\text{where } \gamma_{ki}(t) = \frac{\partial \sigma_k(t_0+h)}{\partial \sigma_i(t)}$$

$$\gamma_{ki}(t) = \begin{cases} \delta_{ki} & \text{if } t = t_0+h \\ g'[\sigma_i(t)] \displaystyle\sum_{l=1}^{N} w_{l(m+i)}\, \gamma_{kl}(t+1) & \text{if } t_0 \leq t < t_0+h \end{cases}$$

Time complexity: $O(N^3)$ per time step, if $h \in O(N)$

Space complexity: $O(N^3)$

Off-line learning: use sum of three terms at the end.

"quasi" on-line learning: use 2nd and 3rd terms at the end of each block of h time steps.