CAPÍTULO 4: ALGORITMOS DE APRENDIZAJE

En este capítulo se proporcionan las descripciones matemáticas de los principales algoritmos de aprendizaje para redes neuronales: *Backpropagation* para las redes *feedforward* y Elman hasta el *Real-Time Learning Algorithm* (RTRL) para las redes recurrentes completamente conectadas, haciendo una comparación de su eficiencia, costo y viabilidad de implementación.

4.1 INTRODUCCIÓN

En ocasiones, se requiere que una red neuronal con varios *Adalines* realice tareas de clasificación más sofisticadas que sólo dividir toda entrada en dos rangos de salidas. En estos casos, se introducen variantes en el diseño general del *Adaline* que mejoren u optimicen dicha respuesta. Entre las formas más comunes de hacer esto está el introducir no-linealidades a la entrada de cada *Adaline* para hacerlos más eficientes. Estas variantes pueden ser funciones polinomiales, cuadráticas, divisiones y multiplicaciones. La estructura general de una unidad de esa naturaleza se muestra en la Figura 4.1 [5,6].

Independientemente del clasificador que se utilice, es necesario establecer un algoritmo que sea responsable de actualizar los pesos de cada elemento del vector W_k, ya que la salida de la red neuronal debe tener la tendencia a eliminar el error en cada iteración. La inclusión de no-linealidades como $\sqrt{X_1}$, $\sqrt{X_2}$ *y* (X_1X_2) modifican los vectores originales (ver Figura 4.1), generando un nuevo vector de entrada al sistema X_1X_2, y también los pesos w_{nk} para producir una salida s_k que pasará por la función de umbral *signum* para generar y_k. Las reglas o algoritmos que rigen este comportamiento pueden ser lineales o no lineales, de un elemento o de más de un elemento. A continuación se presentarán las más importantes.

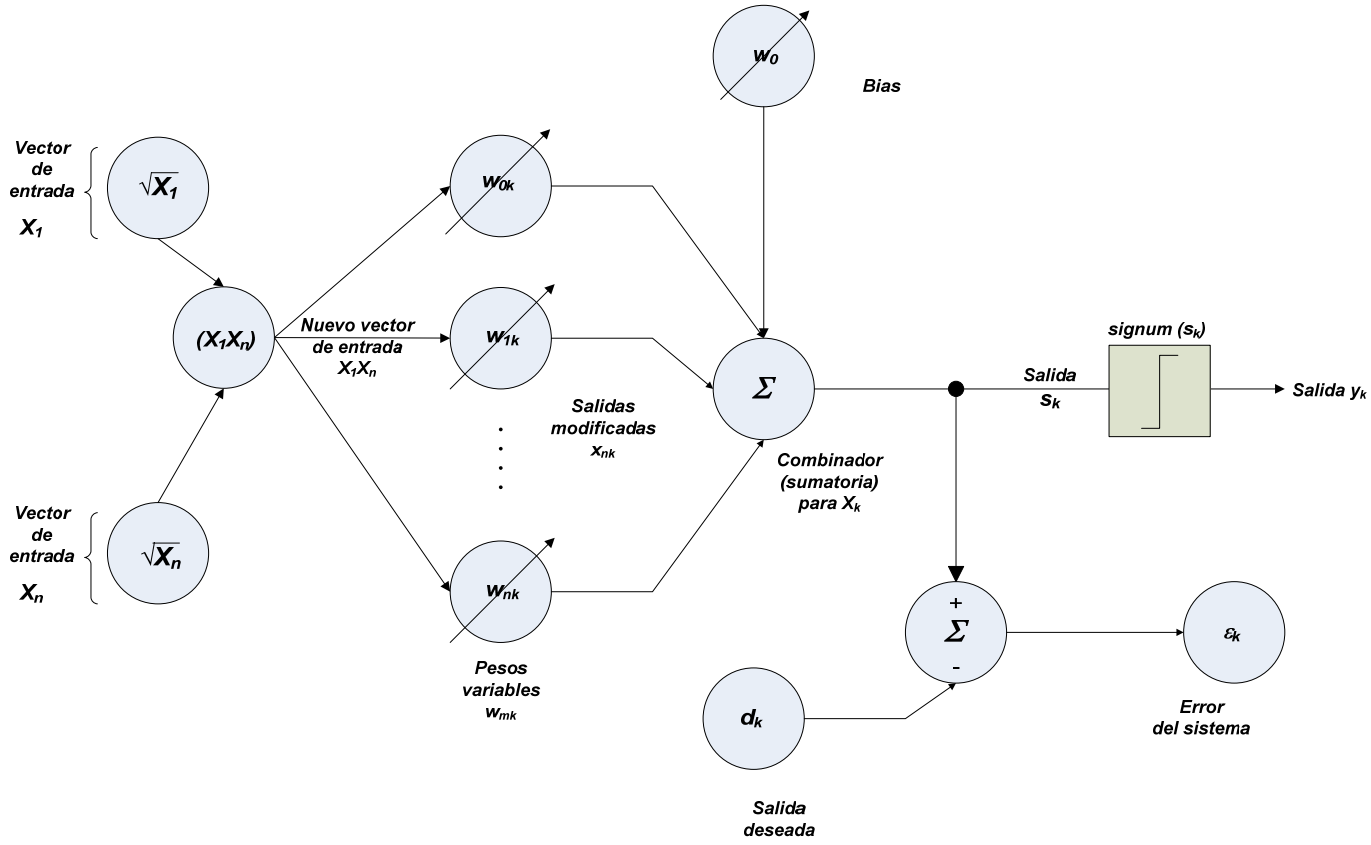

Figura 4.1: Clasificador no-lineal basado en *Adalines* [2]

La primera regla para actualizar los pesos en una red neuronal se conoce como la Regla del Perceptrón, o también el Procedimiento de Convergencia del Perceptrón. Esta primera regla modifica los elementos W_k de acuerdo al algoritmo básico de la Regla del Perceptrón [1]:

$$W_{k+1} = W_k + \alpha \left(\frac{\varepsilon_k}{2} \right) X_k \tag{1}$$

Desarollada por Rosenblatt, esta regla actualiza W_k sólo si el error ε_k es diferente de cero. El vector de entradas es X_k, el nuevo vector de pesos es W_{k+1} y α es la tasa de aprendizaje del sistema (un valor constante muy pequeño que no cambia en el tiempo).

Un poco después, Mays desarrolló su famosa serie de algoritmos adaptativos para redes neuronales, los cuales son una versión más general de la Regla del Perceptrón. Esta nueva regla también es llamada de incremento adaptativo o Algoritmo de Mays [1]:

$$W_{k+1} = \begin{cases} W_k + (\alpha \varepsilon_k)\left(\dfrac{X_k}{2|X_k|^2}\right) & si \ |s_k| \geq \gamma \quad (2) \\ W_k + (\alpha d_k)\left(\dfrac{X_k}{|X_k|^2}\right) & si \ |s_k| < \gamma \quad (3) \end{cases}$$

Donde ε_k es el error $d_k - y_k$, siendo *'d'* la respuesta deseada y *'y'* la salida de la red. Por lo general, la 'zona muerta' delimitada por γ es un valor pequeño, y si vale *0* el algoritmo de Mays se convierte en la Regla del Perceptrón [1].

4.2 EL ALGORITMO *BACKPROPAGATION THROUGH TIME*

Estos algoritmos, no obstante ser útiles en ciertas aplicaciones, deben iterar muchas veces para poder reducir su error. Existe otro método para garantizar un mejor desempeño en términos de convergencia, clasificación y eficiencia: el Método del Descenso Más Rápido. Este método se basa en evaluar el gradiente de la superficie típica de error promedio cuadrado (MSE: *Mean Square Error*) para cada una de las funciones de umbral en diferentes puntos y actualizar los pesos de la red con ese resultado.

$$W_{k+1} = W_k + \mu(-\nabla_k) \quad (4)$$

En esta ecuación, ∇_k es el gradiente en un punto de la superficie MSE. Debemos recordar que ese gradiente en un punto es la derivada parcial del error con respecto al vector de pesos W_k.

$$\nabla_k = \frac{\partial \varepsilon_k^2}{\partial W_k} = \begin{pmatrix} \frac{\partial \varepsilon_k^2}{\partial W_k} \\ \vdots \\ \frac{\partial \varepsilon_k^2}{\partial W_k} \end{pmatrix} \quad (5)$$

Usando estos resultados, es posible definir el algoritmo y reglas que definen el método que se usa para gobernar el comportamiento de redes neuronales que trabajan *off-line*: el conocido como *Backpropagation* [1,3,5].

$$W_{k+1} = W_k + \mu(-\nabla_k) = W_k + \mu \frac{\partial \varepsilon_k^2}{\partial W_k} \quad (6)$$

Como puede verse, la base de este algoritmo es calcular derivadas parciales a través del tiempo en cada iteración por la red neuronal, actualizando los pesos de acuerdo a esto.

Un diagrama general de cómo funciona este tipo de algoritmo se muestra en la Figura 4.2. El vector de entradas X_k va directamente a las neuronas y también al vector W_k para modificar los pesos w_1 a w_n. Dichos pesos también son modificados con el resultado de la salida y_k tras pasar por el combinador y la influencia del elemento constante con su peso w0, la cual es multiplicada por la tasa constante de aprendizaje 2m y alimentada al algoritmo LMS (*Least Mean Square*) para calcular el error que debe retroalimentarse a la actualización del vector de pesos W_k a través de un proceso iterativo [1].

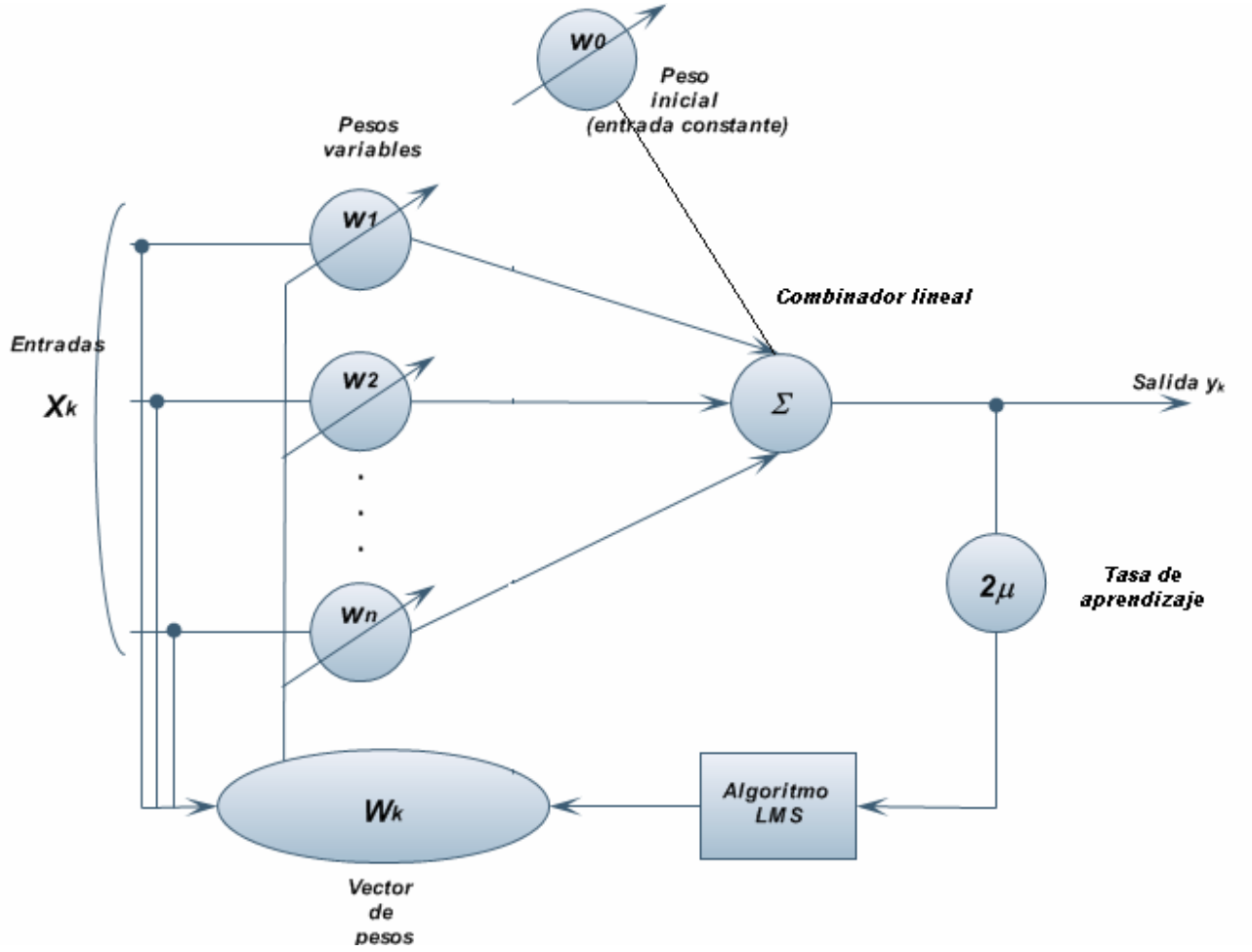

Figura 4.2. Red retroalimentada por *backpropagation*

4.3. EL ALGORITMO DE APRENDIZAJE EN TIEMPO REAL

Una red neuronal recurrente en el tiempo tiene la capacidad de calcular la siguiente salida en base a los resultados preliminares de sus salidas anteriores (ver Figura 3.1), a diferencia de una red *feedforward* la cual solamente puede realizar este proceso hasta cierto punto en el tiempo antes de detenerse (Figura 3.2).

$$y(t) = \Phi\left(x(t), x(t-1), \ldots x(1), x(0)\right) \tag{7}$$

$$y(t) = \Phi\left(x(t), x(t-1), \ldots x(t-h+1)\right) \tag{8}$$

El algoritmo que gobierna el comportamiento de las RNN en tiempo real, *Real-Time Recurrent Learning Algorithm* o RTRL desarrollado por William y Zipser [5], tiene una complejidad de cálculo para un número de unidades procesadoras directamente conectadas a la red de $O(n^4)$, lo cual es bastante elevado e implica un gran número de cálculos a mayor número de elementos, pero tiene una alta probabilidad de convergencia [13,14].

La activación de cada nodo para cada instante *t*, s_k, está definida por:

$$s_k(t) = \sum_{p \in I} w_{kp} x_p(t) + \sum_{q \in U} w_{kq} y_q(t) \qquad k \in U \tag{10}$$

Aquí, *U* es el conjunto de índices de los nodos procesadores, *I* es el conjunto de índices de las entradas del sistema, y_q es una salida previa y $w_{kp,q}$ representa los pesos conectando nodos. La salida sk del nodo k es función de f_k, una función sigmoide.

$$y_k(t+1) = f_k\left(s_k(t)\right) \tag{11}$$

El error cuadrático a cada instante de tiempo *t* es igual al cuadrado del error $e_k(t)$.

$$E(t) = \sum_{k \in U} e_k^2(t) = \frac{1}{2} \sum_{k \in T(t)} \left\{d_k(t) - y_k(t)\right\}^2 \tag{12}$$

Sólo si e_k pertenece al conjunto de índices que engloba *U*. De otra manera, e_k es igual a cero. Los pesos, por su parte, se actualizan mediante la regla del descenso por gradiente discutida anteriormente, en donde α es el *learning rate,* tasa de aprendizaje, de la red e *I* es el conjunto de índices correspondientes a la entrada del sistema.

$$\begin{aligned}\Delta w_{ij}(t) &= -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \\ &= \alpha \sum_{k \in U} e_k(t)\, p_{ij}^k(t) \qquad\qquad i \in U, j \in U \bigcup I\end{aligned} \tag{14}$$

El término *p* es denominado la sensibilidad del nodo k en cuestión a un cambio en el peso w_{ij} [3].

$$\begin{aligned}p_{ij}^k(t+1) = \quad & f_k'(s_k(t)) \left\{ z_j(t) \delta_{ki} + \sum_{q \in U} w_{kq} p_{ij}^q(t) \right\} \\ & i,k \in U, j \in U \bigcup I\end{aligned} \tag{15}$$

En donde δ_{ki} es la delta de Kronecker (con valor de 1 si los subíndices i,j son iguales y 0 si son diferentes), y z_j es la salida y_j. f'_k denota la derivada de la función sigmoide *f*. El algoritmo RTRL está definido por todas estas ecuaciones, y su puesta en práctica depende de su correcta implementación [1,3,5,14,15].

Como los algoritmos de entrenamiento tales como el RTRL y el BPTT se basan en el gradiente de una función de costo que se relaciona directamente con los pesos de toda la red, es deseable considerar las ventajas y desventajas de cada uno para justificar el uso, en este caso del RTRL.

La demanda en términos computacionales del algoritmo RTRL es mayor a la del BPTT. Sin embargo, esta característica negativa es compensada por el hecho de que, usualmente, las redes neuronales basadas en RTRL requieren de una cantidad de neuronas significativamente menor que sus contrapartes. Además, el RTRL es un algoritmo capaz de trabajar en modo *on-line* y esto hace que para ciertas aplicaciones, tales como un sistema IDS, sea necesario utilizar un algoritmo de este tipo [13].

La diferencia fundamental entre ambos algoritmos, y lo que explica el hecho de que el BPTT esté limitado a trabajar *off-line*, es el cálculo del gradiente y cómo es realizado, ya sea hacia atrás en el tiempo como con el algoritmo BPTT, o hacia delante como el RTRL. Esta característica permite que los parámetros de entrada para este último tengan un efecto transitorio en la salida general de la red mientras que los pesos cambian constantemente guardando un 'recuerdo' de las condiciones iniciales que afecta su actualización iterativa hasta que la red ha terminado su proceso de entrenamiento.

4.4 RESUMEN

A través del desarrollo e investigación del comportamiento de las redes neuronales se han desarrollado muchos algoritmos para acelerar su aprendizaje y reducir el tiempo que le toma a la red alcanzar el estado interno deseado. Desde el desarrollo de los primeros algoritmos de actualización de pesos basados en las investigaciones de Mays y otros se ha podido incrementar la efectividad del entrenamiento hasta el punto en el que las redes actuales sobrepasan por mucho a las primeras.

Uno de los algoritmos más utilizados actualmente es el conocido como BPTT. Utilizado en una gran variedad de aplicaciones, su rapidez relativa lo hace atractivo

como opción cuando la complejidad de la red es tal que un algoritmo más estructurado ocasionaría retardos enormes de ser aplicado.

Como su nombre lo indica, este algoritmo propaga la información concerniente al gradiente y los valores de actualización de los pesos hacia atrás en el tiempo, utilizando los resultados actuales para modificar a la red antes de que se ingrese una nueva secuencia de entrada [21]. Este algoritmo produce buenos resultados en aquellas aplicaciones que no tienen una relación secuencial fuerte y por su naturaleza son eventos independientes unos de otros. Por esto, este tipo de redes trabajan *off-line*, modificando el valor de sus estados internos en tiempos relativos al contexto en el que están siendo utilizadas.

Por otro lado, el algoritmo de aprendizaje en tiempo real permite a las redes predecir el siguiente evento basadas en la información previa. Para estas redes, todos sus estados anteriores son accesibles gracias a que el algoritmo actualiza los pesos utilizando cada estado previo. El RTRL permite a las redes neuronales trabajar en forma *on-line*, actualizando sus valores en tiempo real y arrojando resultados satisfactorios que se adapten a las condiciones cambiantes del medio. Para un IDS, este tipo de algoritmo es la mejor opción por todas las características ya mencionadas, y el único inconveniente de elegir el RTRL como opción es precisamente el hecho de que todo ese procesamiento afecta la velocidad de respuesta del sistema, siendo ésta $O(n^4)$, donde n es el número de neuronas.

Puede verse con facilidad que, para sistemas muy grandes, el procesamiento tiende a ser lento; no obstante, para aplicaciones con pocas neuronas el RTRL arroja resultados muy buenos [16].