CAPÍTULO 5: MODELADO DEL IDS CON REDES NEURONALES

En este capítulo se describe la preparación de los datos para servir como entradas al IDS y la simulación de las redes Recurrentes completamente conectadas, Elman y *feedforward* utilizadas a través de su entrenamiento con los algoritmos de *Backpropagation* y el *Real-Time Learning Algorithm*. El código en MATLAB® correspondiente puede encontrarse en el Apéndice A.

5.1 FORMATO DE LOS DATOS

Las entradas al sistema detector de intrusos no pueden tener cualquier valor. Cada aplicación a la que son sometidas las redes neuronales requiere de ciertas rutinas de pre-procesamiento de datos para que éstos sean adecuados para su análisis por dicha red. Por ejemplo, una aplicación en la que se requiere que la red identifique imágenes y decida si coinciden con una imagen maestra o no y otra en donde tenga que diferenciar entre listas de comandos para clasificarlos como ataques o instrucciones normales necesitan rutinas que conviertan las entradas (imágenes, cadenas de caracteres) en algo que la red neuronal pueda entender.

El formato a utilizar es el formato binario: para esta aplicación en particular, las bases de datos con comandos en HTTP representativos de las categorías normal, XSS, SQL, *path* e inyección deberán ser transformados de cadenas de caracteres alfanuméricos en cadenas de caracteres binarios.

Cada una de las cadenas de caracteres corresponde a un tipo de comando normal, XSS, SQL, de *path* o inyección respectivamente. Es evidente que tienen longitud variable y que los caracteres no están en formato binario, por lo que se debe procesar toda esta información para que la red neuronal pueda trabajar con ella [9].

Tabla 5.1: Ejemplo de datos de entrada

CADENA DE ENTRADA	*CLASIFICACIÓN*
&@_@=@&@_@	NORMAL
")</@>/@.@	XSS
' --@	SQL
". /.". /@/@	PATH
&@=";@ ";	INJECTION

El primer paso es convertir todos esos caracteres a su equivalente ASCII. Cada cadena de caracteres se convierte en una cadena de números decimales que es más fácil de manipular, y la cual puede ser transformada a formato binario en un paso posterior. Tomando como ejemplo las cadenas que presentamos hace poco, su representación como números decimales ASCII es la siguiente:

Tabla 5.2: Conversión a formato ASCII

CADENA ASCII	*CLASIFICACIÓN*
38 64 95 64 61 38 64 95 64	NORMAL
34 41 60 47 64 62 47 64 45 64	XSS
39 45 45 64	SQL
34 46 47 46 34 46 47 64 47 64	PATH
38 64 61 34 59 64 61 34	INJECTION

Para solucionar el problema de longitud variable definimos un vector de tamaño fijo como entrada de tamaño ocho. Este tamaño es lo suficientemente grande como para evitar excesivas iteraciones de procesamiento y lo suficientemente pequeño para omitir

el llenar los espacios vacíos en cadenas menores a ocho con el número 32 (en ASCII, espacio en blanco). Si una cadena de caracteres queda expresada de la siguiente forma:

34 41 60 47 64 62 47 64 45 64

Al someter estos datos a la rutina de ventana deslizante que requiere un vector fijo de longitud N = 8, se genera:

34 41 60 47 64 62 47 64

41 60 47 64 62 47 64 45

60 47 64 62 47 64 45 64

Es importante mencionar que al pasar los datos de entrada por la rutina de ventana deslizante, en realidad se generan más vectores de entrenamiento de los que se tenían originalmente. Esto lejos de ser una desventaja en un beneficio para la red, la cual puede estar expuesta a más patrones de entrenamiento antes de ser probada.

Una vez que se tienen los vectores con tamaño fijo es momento de transformarlos a formato binario. Esto es necesario por varias razones. En primer lugar, nuestra red neuronal requiere estos valores porque son una forma homogénea de presentar las cadenas de caracteres que originalmente eran números ASCII con formato decimal, los cuales necesitarían mayor tiempo de procesamiento de mandarse las cadenas tal cual son.

En segundo lugar, la mayoría de las funciones de umbral que se utilizan a la salida de las neuronas son funciones cuyos valores de mayor pendiente se encuentran en el rango de cero a uno. Las funciones sigmoideas que se utilizaron en esta aplicación, por ejemplo, tienen su rango de mayor pendiente entre -1 y 1; más allá de esos valores

la pendiente se vuelve muy pequeña y es evidente que, no importando qué tan grandes sean los cambios en los valores de los vectores de entrada, si la función de umbral no puede arrojar un resultado lo suficientemente diferente entonces la red no reconocería dos valores radicalmente distintos y su desempeño se vería negativamente afectado. La función tangente hiperbólica, en particular, fue la función de umbral sigmoidea seleccionada y cumple con todos estos requerimientos (ver Figura 5.1).

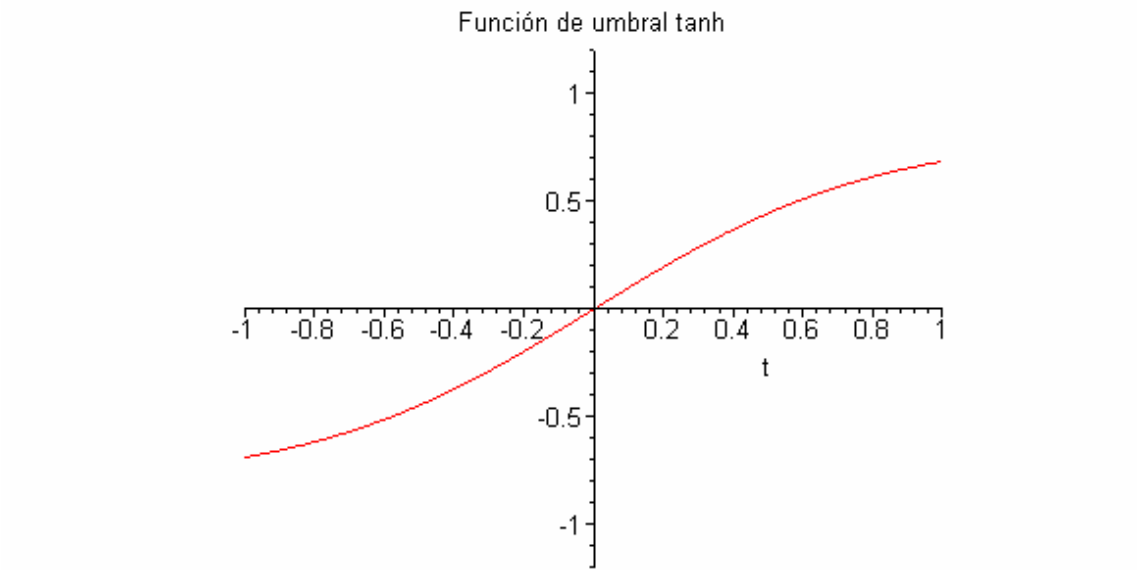

Figura 5.1. Función de umbral (tangente hiperbólica)

Con los datos transformados a formato binario, las cadenas de caracteres originales con tamaño fijo N=8 se transforman, cada una, en cadenas de longitud N = 64. Esto significa que nuestro vector de entradas será de 64 elementos binarios, y por ende la primera capa de la RNN, la capa de *inputs*, será de 64 neuronas para que cada una proceso uno de los elementos binarios presentes en los datos de entrada originales en paralelo [10].

5.2 MODELADO DEL IDS

Ya que los datos han sido preparados, es momento de diseñar las redes neuronales a utilizarse para entrenarlas y probarlas con dichos datos. El primer problema es hallar el número de neuronas que cada una debe poseer, el número de capas de ocultas (si hay) en ellas y el número de neuronas de salida.

El número de datos de entrada se determinó con base a la ecuación usado por Williams y Zipser [14], $N \geq W/\varepsilon$, en donde W es el número total de pesos presentes en la red y ε es un parámetro constante de error cuyo valor característico es de 1%. De esta manera administraremos nuestro conjunto de datos de entrada para lograr resultados con márgenes de error cercanos al uno por ciento.

En cuanto a las neuronas de salida, está claro que nosotros queremos que el IDS mande una señal binaria que pueda ser reconocida por un dispositivo complementario para tomar medidas correctivas en caso de un ataque a la red. Para simplificar esta señalización lo más posible, el número de neuronas en la capa de salida debe ser igual al número de tipos de comportamiento normal y peligroso que estamos considerando: en este caso, cinco tipos correspondientes a las categorías Normal, Inyección, *Path*, SQL y XSS. Así, cuando se identifique una cadena de caracteres como perteneciente a un comando Normal, la neurona 1 del sistema adquirirá un valor binario de 1 mientras que las demás permanecerán en 0. De ser un ataque, si por ejemplo la neurona 5 está en 1, debe entenderse que fue un ataque tipo XSS. También sería posible diseñar la red con únicamente dos neuronas en la capa de salida, con categorías Normal y Ataque, pero entonces perderíamos claridad en la identificación y solamente sabríamos que un ataque tuvo lugar, pero no qué tipo de ataque. Por todas estas razones, cinco neuronas en la capa de salida de la red ha resultado ser el número ideal.

Con respecto al número de neuronas en las capas ocultas, esto depende de qué tipo de red se quiera modelar. Nuestra primera prueba, una red *feedforward* simple, fue diseñada con un tamaño de 64x15x15x5. La red se verá entonces como en la Figura 5.2, con 64 neuronas en la capa inicial, 15 en las dos capas ocultas y 5 neuronas del vector y_k a la salida correspondiendo cada una a un tipo de ataque:

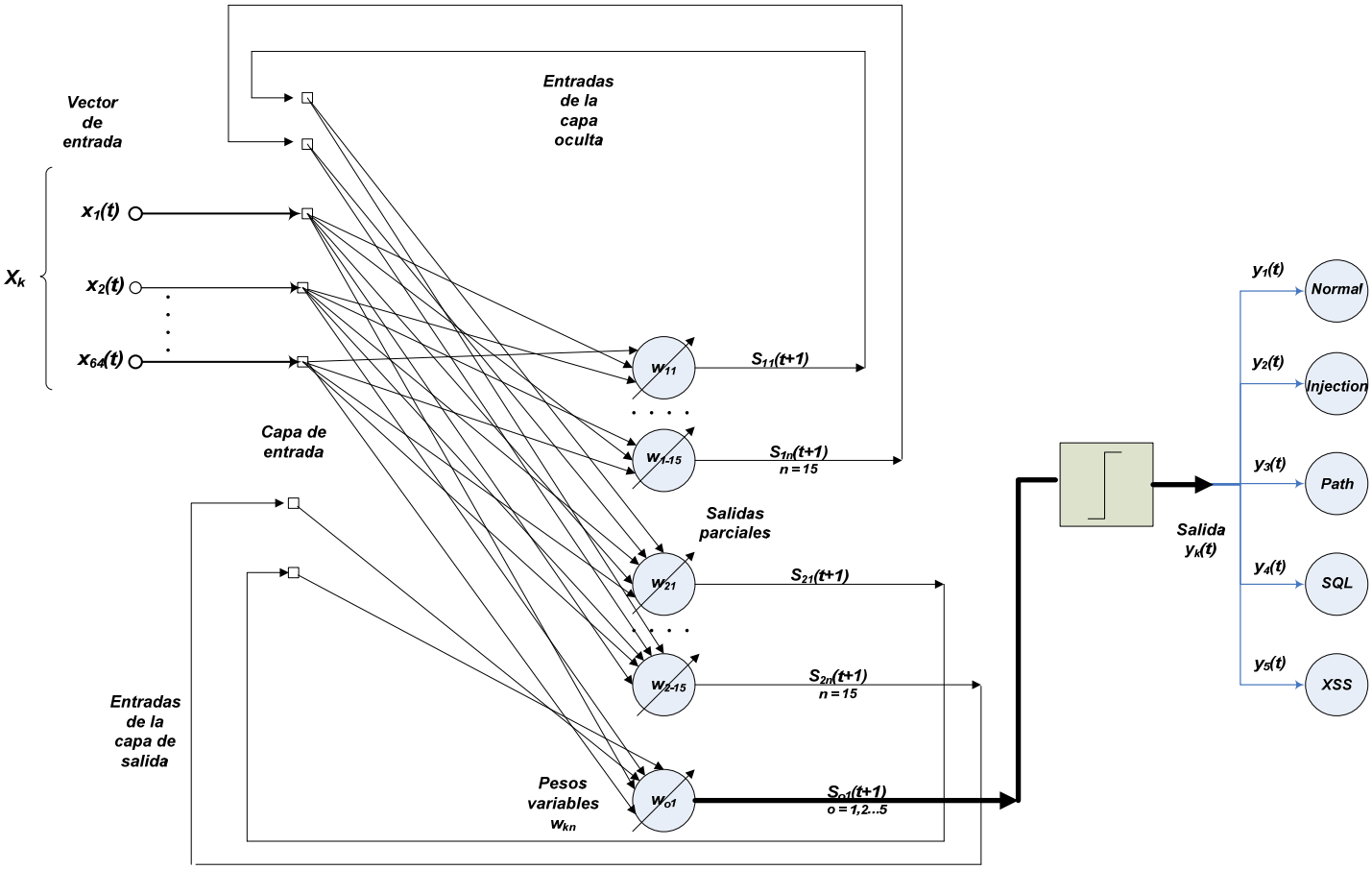

Figura 5.2. Red *feedforward*, 64x15x15x5

Como se explicó en el capítulo 3, todas las neuronas de una capa están interconectadas con las neuronas de la siguiente, iniciando con la capa principal y los elementos del vector X_k, proporcionando su información (las salidas $s_{mn}(t+1)$) hacia delante dentro de la red. El vector de pesos W_k es actualizado conforme los *epochs* pasan y al final del último los pesos individuales $w_{11}…w_{1-15}$, $w_{21}…w_{2-15}$, etc, asumirán sus valores finales una vez que se ha llegado a cinco salidas globales son $(t+1)$ que

pasarán por la función de umbral *signum.* Un *epoch* es un ciclo completo de procesamiento en la red, desde la entrada hasta la salida para un *input*nuevo.

Los criterios para definir el número de neuronas en las capas ocultas fueron los siguientes. En primer lugar, ciertos autores [13, 14] sostienen el punto de vista en el que el número máximo de capas que una red neuronal debe tener sin incurrir en sobre-procesamiento de los datos es dos. Más allá de este número de capas, la red exhibe un buen comportamiento en problemas de muchas variables y también en no-linealidades a través del tiempo dependientes de su función de umbral, pero estas características sólo son deseables en algunos de los problemas; la vasta mayoría de ellos no requiere tal cantidad de neuronas. En segundo lugar, el número mismo de neuronas por capa para una red *feedforward* como la que tenemos aquí ha sido definido por (Ni - No) /4 [13], donde Ni es el número de neuronas a la entrada y No es el número de neuronas a la salida de la red, respectivamente. En este caso, con Ni = 64 y No = 5 nuestras capas ocultas deberán tener (redondeando al entero inmediato superior) 15 neuronas. Esta es la razón por la que la red *feedforward* fue diseñada así.

Para la segunda red analizada como opción para ser la base de un IDS, elegimos una SNR Elman. Estas redes cuentan con caminos de retroalimentación como se mencionó en el capítulo 3, lo cual hace que requieran un mayor número de neuronas para poder aprovechar adecuadamente su capacidad de recordar el estado interno inmediato anterior de la red. Una red Elman con el mismo número de neuronas en las capas ocultas que una red *feedforward* tendría muy poca diferencia en cuanto a desempeño. No obstante, si se le diseña apropiadamente puede sobrepasarla con facilidad en velocidad de convergencia y porcentajes de error.

En este caso, el criterio para diseñarla fue, de nuevo, el que el número máximo de capas ocultas sin incurrir en procesamiento no necesario son dos. Para hallar el

número de neuronas necesarias en cada una de las capas ocultas recurrimos a la fórmula siguiente: (Ni – No) / 2 [13], en donde las variables tienen el mismo significado que para la red *feedforward*. Para nuestra aplicación basada en HTTP, esto nos da un resultado aproximado de 30 neuronas por capa (ver Figura 5.3). Esto es el doble de neuronas que las de la red anterior, pero se espera que este costo adicional se compense con la rapidez y exactitud del desempeño de la red. La notación en la figura es similar a la de la red *feedforward* (ver Figura 5.2), pero las interconexiones son distintas.

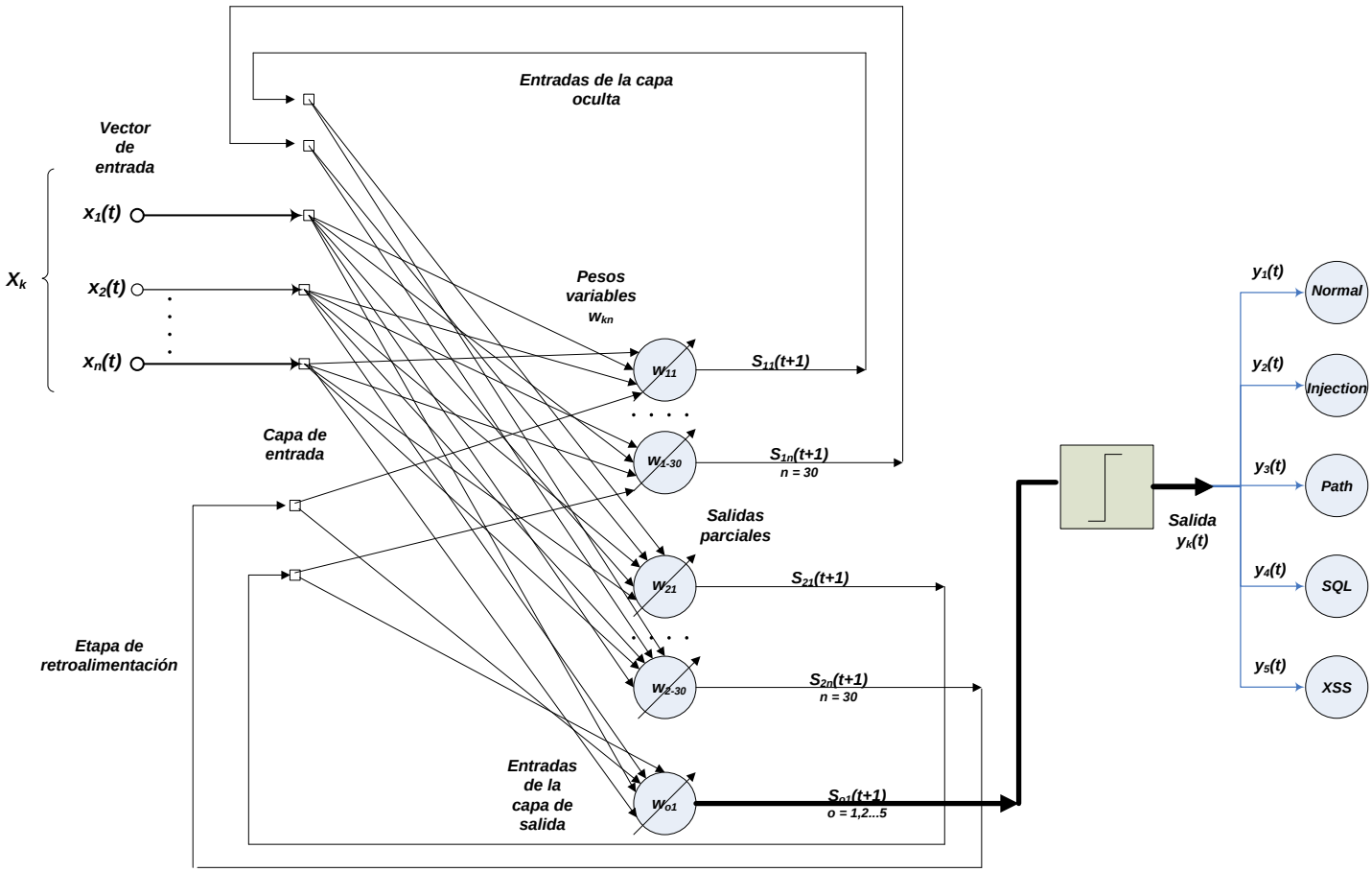

Figura 5.3. Red Elman 64x30x30x5

La última red diseñada, y la que es la base de este proyecto, es la RNN completamente conectada. Este tipo de redes tienen índices de convergencia más elevados que las demás y su porcentaje de error es por lo general muy bajo.

Evidentemente, la complejidad de las interconexiones entre capa y capa tienen como consecuencia tiempos de procesamientos del orden de $O(n^4)$, como se mencionó en el capítulo 4, utilizando el algoritmo de aprendizaje en tiempo real. Por esta razón, es muy importante hallar el tamaño exacto de la red y no excederse en el número de neuronas a utilizar o en el número de capas ocultas para poder tener todas las ventajas de una RNN sin el costo tan alto de procesamiento [1,3].

La estructura de la red utilizada para el IDS es mostrada en la Figura 5.4, con los 30 elementos en la única capa oculta, los *delays* a la salida de dicha capa y la clasificación final, con la misma notación que las figuras anteriores.

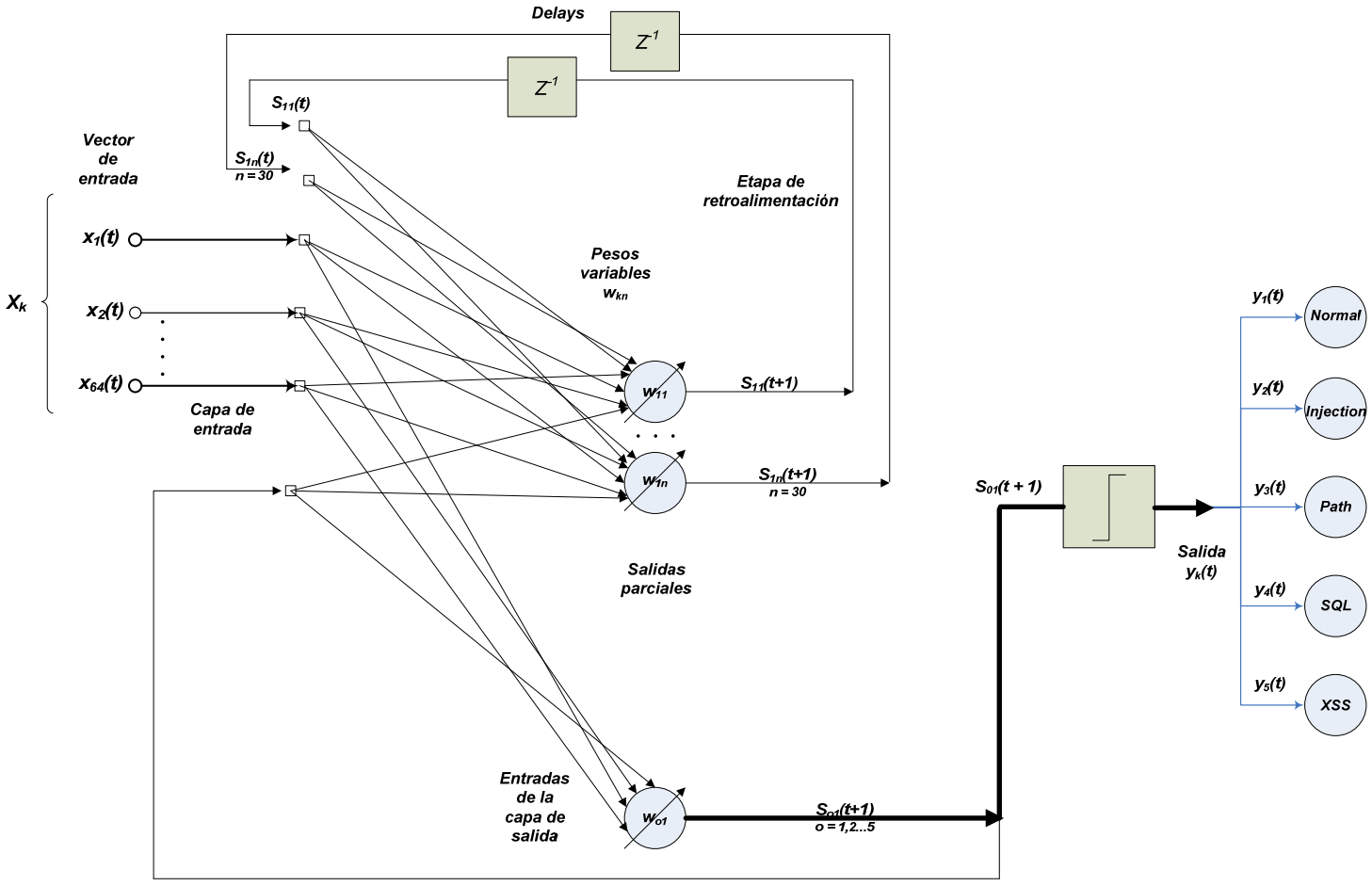

Figura 5.4. RNN completamente conectada 64x30x5

El número de capas recomendado para una RNN [13], a diferencia de las otras dos, es solamente una. Agregar una segunda capa mejora el desempeño de la red, es verdad, pero dicha mejora es pequeña y para un problema de aplicación como un IDS en el que pequeños márgenes de error son tolerables la mejor opción es trabajar con una única capa oculta. El solo hecho de que una RNN sobrepase en exactitud a una SNR Elman o a una red *feedforward* utilizando sólo una capa oculta es indicativo de las grandes ventajas que este tipo de diseño tienen sobre los demás. Finalmente, para calcular el número de neuronas por capa seguimos la fórmula (sección 5.1) anterior aplicada a la red Elman y obtenemos 30 neuronas en la capa oculta.

5.3 SIMULACIÓN DE LAS REDES NEURONALES

Cuando los parámetros de diseñado fueron determinados, el siguiente paso a seguir fue entrenar las redes por medio de simulación en MATLAB ®. Para esto, se fijó un valor de máximo error permitido y, utilizando los algoritmos BPTT y RTRL, se entrenaron las redes para determinar en qué momento alcanzarían el valor esperando de error y con ello demostrar que los valores de los pesos en cada una de las capas fueran óptimos para iniciar la sesión de prueba.

La primera red que probamos fue la red *feedforward*. Para entrenarla utilizamos una variante del algoritmo BPTT disponible en el ToolBox de MATLAB® para redes neuronales, propagación elástica. La variante de este algoritmo con respecto al BPTT tradicional es únicamente el hecho de que, al calcular el gradiente a través del entrenamiento de la red, se toma únicamente el signo de dicho parámetro para determinar si la superficie de error crece o decrece y así decidir si los pesos deben seguir cambiando con la pendiente actual o cambiarla en caso de acercarse al mínimo global [10].

Tras la simulación, el código fuente de la cual está disponible en el apéndice anexado a este trabajo, el desempeño de la red *feedforward* fue el siguiente:

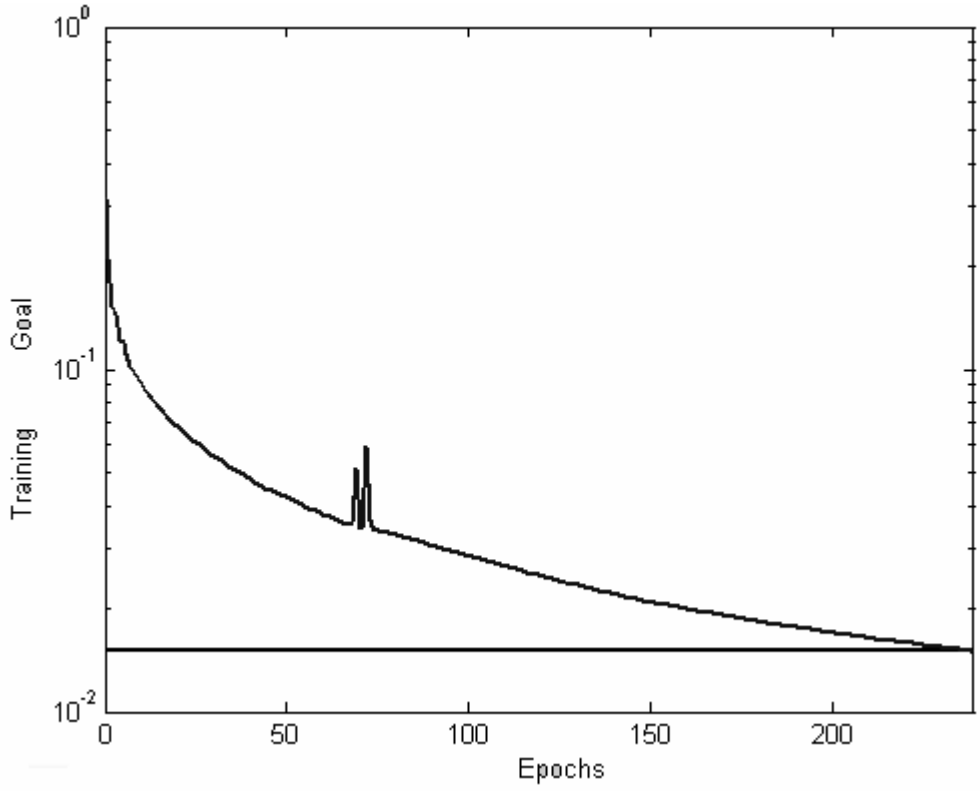

Figura 5.5. Simulación de la red *feedforward*, 64x15x15x5

De acuerdo a la gráfica, la red alcanza el valor máximo de error permitido (0.015) en un total de 239 *epochs* (Figura 5.5). Hasta el momento no contamos con ningún parámetro de comparación, por lo que debemos esperar hasta simular las otras dos redes para ver si este valor es bueno o malo.

La segunda red, la SNR Elman, fue simulada utilizando el mismo algoritmo de entrenamiento que la red *feedforward* disponible en MATLAB®. De esta manera fue posible apreciar cómo ambas arquitecturas arrojaban resultados diferentes a pesar de tener las mismas condiciones iniciales en términos del número y clase de datos a la

entrada al iniciar el entrenamiento y también el método de aprendizaje con esta información.

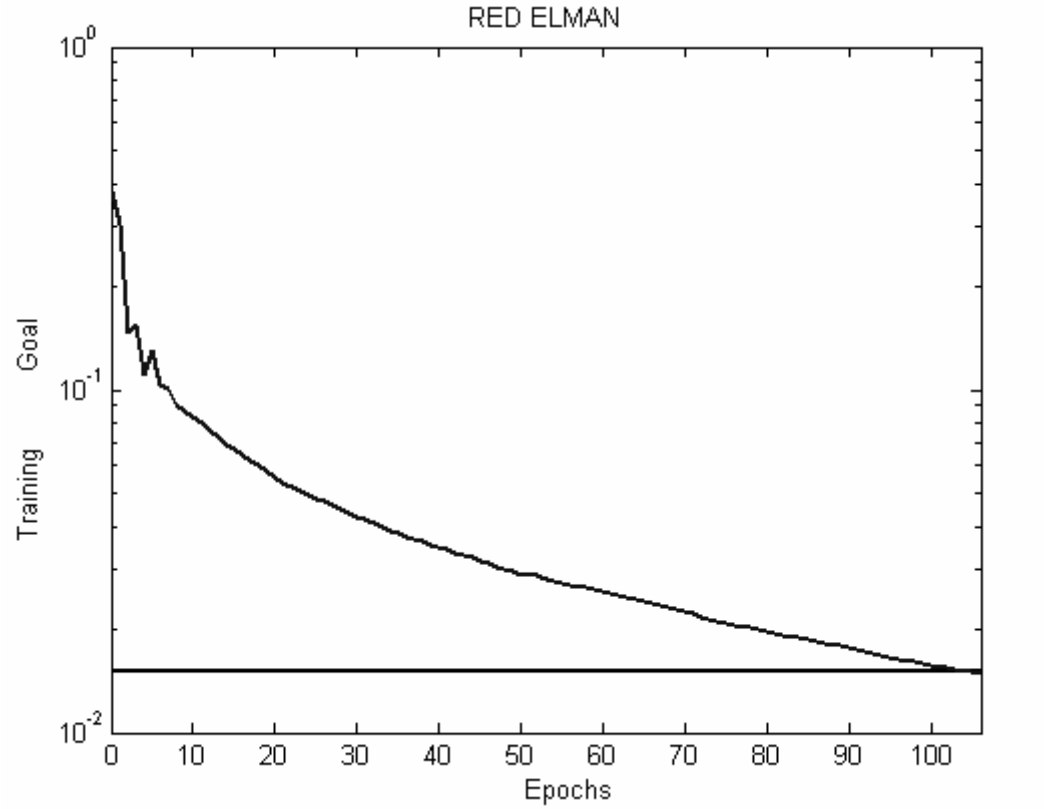

Figura 5.6. Simulación de la SNR Elman 64x30x30x5

Al comparar estos resultados (Figura 5.6) con los de la simulación de la red *feedforward*, vemos que hubo un incremento dramático en la velocidad con la que la red se estabilizó en el margen de error permitido debido a los requerimientos de un sistema de detección de intrusos. A comparación de los 239 *epochs* de la simulación anterior, esta red logró la meta en sólo 106, lo que representa una mejora importante de desempeño acorde con lo que esperábamos de esta red neuronal recurrente simple. Podemos ver que el haber aumentado el número de neuronas en las capas ocultas tuvo el efecto predicho, a pesar de ciertas pequeñas oscilaciones indeseables cerca de los 60

epochs que son parte del comportamiento de estas redes, reaccionando a los máximos o mínimos locales antes de estabilizarse en los valores requeridos.

La última red, la que se ha considerado como la mejor opción para el IDS a diseñar, es la RNN completamente conectada. Ya se ha mencionado en la sección anterior que esta red cuenta con sólo la mitad de capas ocultas que las demás redes, pero sus capacidades de procesamiento, generalización y predicción a través del tiempo la hacen más efectiva que cualquiera de las otras.

Para entrenar esta red se utilizó el algoritmo RTRL descrito en la sección 4.3. Una diferencia muy importante con respecto a las otras dos simulaciones es que en este caso MATLAB® no cuenta con funciones especializadas dentro de su *ToolBox* de redes neuronales para entrenar a las redes automáticamente. En los dos casos anteriores bastó con aprender la sintaxis de dichas funciones para realizar el entrenamiento, pero para este último caso gran parte del tiempo del proyecto se dedicó precisamente a desarrollar las rutinas de creación, entrenamiento y prueba para el RTRL utilizando funciones propias y adaptándolas al formato y tamaño de los datos que utilizaron las otras dos redes neuronales. El código fuente de todas las funciones desarrolladas para llevar a cabo este entrenamiento está disponible en el apéndice adjunto.

La gráfica del desempeño de la última red confirmó las expectativas en cuanto a su funcionamiento, ya que demuestra ser más rápida y alcanzar su resultado en menos iteraciones (Figura 5.7):

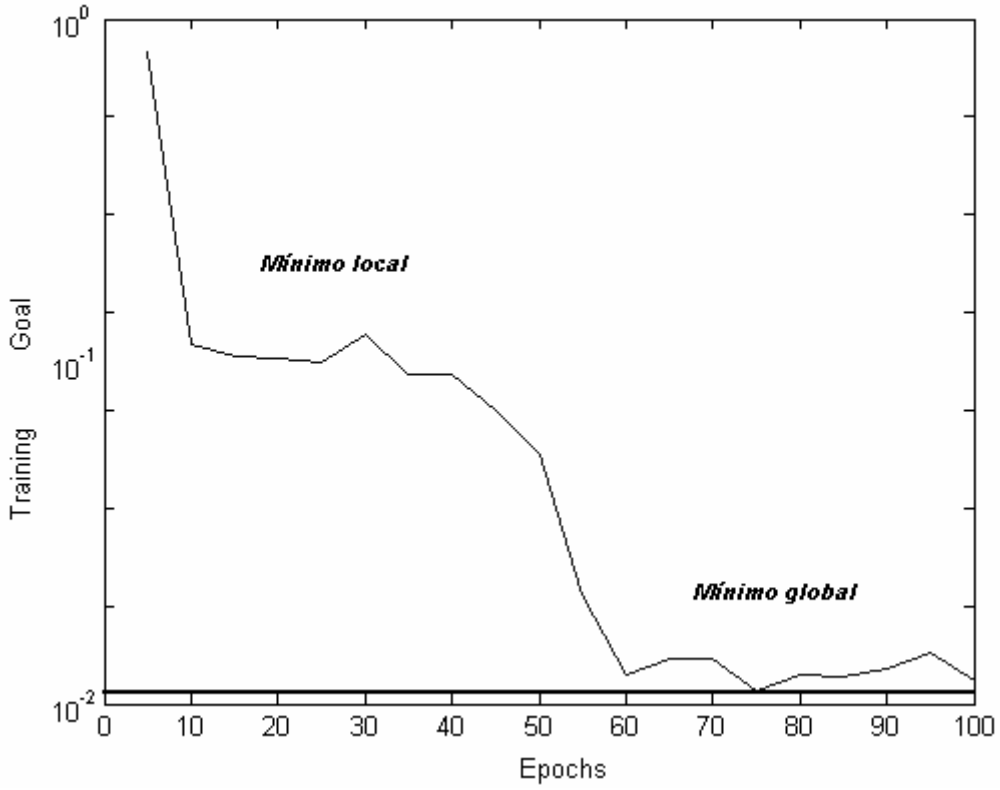

Figura 5.7. RNN completamente conectada 64x30x5

Como puede apreciarse en la gráfica, esta red alcanzó el parámetro de error esperado en un número de *epochs* significativamente menor a las otras dos: únicamente 60 *epochs* comparados con 106 y 239 respectivamente. Su poca variación en el intervalo 10 – 40 *epochs* se debe a que el RTRL ha encontrado un mínimo local, pero al no ser el valor deseado (0.38 vs 0.015), no es el valor permanente de la red y luego se encuentra el verdadero mínimo al llegar a 60 epochs, después de lo cual sólo hay variaciones pequeñas en la salida de la red. A pesar de tener sólo una capa oculta obtuvimos una mejora en velocidad de alrededor del 40%, lo cual concuerda perfectamente con nuestras expectativas iniciales.

5.4. RESUMEN

La parte inicial de todo proceso de simulación utilizando redes neuronales es dar un formato adecuado a los datos. Una vez que los datos se encuentran en formato binario a través de sucesivas transformaciones llevadas a cabo por rutinas que procesen la información y la adecuen a las capacidades de la red, es cuando es posible empezar a simular las redes neuronales con sus entradas adecuadas.

Las tres arquitecturas opcionales de redes neuronales fueron modeladas siguiente las guías de diseño sugeridas por varios autores. Con base en estas premisas fue posible determinar el número de capas ocultas que conformarían cada red para garantizar un desempeño bueno sin incurrir en procesamiento excesivo, el número mismo de neuronas por capa oculta para agilizar el tiempo de convergencia de cada sistema y la forma en que estos parámetros de diseño interactuarían con los algoritmos de entrenamiento discutidos en secciones anteriores.

Evaluando la respuesta al entrenamiento de cada red, la RNN demostró claramente ser la más rápida en alcanzar un error aceptable y estabilizarse consecutivamente en ese valor, siendo 40% más rápida que la SNR Elman y 75% más veloz que la red *feedforward.* Estos resultados son muy promisorios pero representan únicamente la respuesta del sistema al entrenamiento. Los verdaderos parámetros de evaluación serán los resultados que arroje cada red al sometérsele a entradas nuevas con fines de prueba, y verificar que el IDS reconozca y clasifique ataques como debe.