CAPITULO 6 SISTEMA DE DETECCION DE INTRUSOS USANDO REDES NEURONALES.

En este capítulo se realiza la preparación adecuada de toda la información para el desarrollo final del sistema de detección de intrusos para redes de comunicaciones con redes neuronales.

6.1 PREPARACION DE LOS DATOS.

Antes de comenzar el diseño es importante realizar la preparación consistente de los datos para que tengan el formato necesario para servir como información de entrada de una red neuronal. Para esto se ha realizado tres pasos que se describirán a continuación.

6.1.1 CONVERSION A FORMATO DECIMAL.

En primer lugar las cadenas de caracteres, que actualmente se encuentran almacenadas en este formato, deben ser convertidas a un valor decimal, de tal forma que obtengamos una nueva cadena de valores ASCII correspondientes a cada una de las letras de la cadena de caracteres original. Por ejemplo, la siguiente cadena que representa un ataque de inyección de comandos *'@.cgi?@=%0a@'*, queda representada de la siguiente forma:

[64 46 99 103 105 63 64 61 37 48 97 64]

De esta manera todas las cadenas de caracteres pueden ser manipuladas como datos decimales que tengan significado numérico [TOR03].

6.1.2 VENTANA DESLIZANTE.

Posteriormente, debemos darnos cuenta que las cadenas decimales con las que contamos, suelen tener un tamaño variable, es decir que algunas tienen 5 elementos, otras

12 elementos y en general la gran mayoría tiene un número variable de elementos. Si se pretende que estas cadenas sean entradas de entrenamiento para nuestra red neuronal, debemos considerar el hecho de que nuestra red neuronal posee un número de entradas fija, una vez que ha sido definido. De manera que si tiene un número fijo de 8 entradas no podrá procesar una cadena de 12 elementos.

Para solucionar este problema, hemos diseñado una ventana deslizante, cuya labor es la de convertir las cadenas de tamaño variable, en cadenas de tamaño fijo, de manera que nosotros fijemos el número estático al que se deberá ajustar la longitud de todas las cadenas con las que contemos. Para esto imaginemos que poseemos las siguientes cadenas decimales:

120 33 45 64 32

23 34 45 67

88 90 14 57 10 13 45

24 15 12

Sabiendo que la cadena más chica tiene un tamaño de tres, se puede tomar este número como el tamaño fijo que deberán tener todas las cadenas. Así pues las cadenas anteriores quedarían representadas de la siguiente manera:

120 33 45

33 45 64

45 64 32

23 34 45

34 45 67

88 90 14

90 14 57

14 57 10

57 10 13

10 13 45

24 15 12

De esta manera se han generado cadenas de tamaño fijo al costo de generar más cadenas de entrenamiento, costo que en realidad es un beneficio extra al permitir a la red observar más patrones de entrenamiento con que poder generalizar [TOR03]. Este proceso queda más claro en la Figura 6.1 que se muestra a continuación, cuando se quiere pasar por la cadena deslizante el vector [36 98 27 53 54 66].

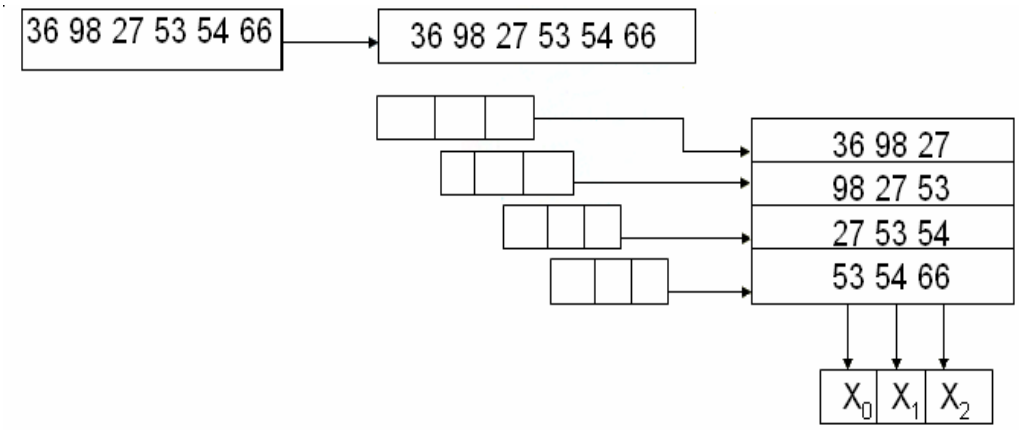

Figura 6.1 Ventana deslizante.

Cabe mencionar que el programa desarrollado cuyo código se encuentra en el código fuente del apéndice A, ha sido diseñado para realizar la ventana deslizante del tamaño que sea, sin importar si este tamaño es mayor que el tamaño de la cadena de menor longitud. De manera que los huecos faltantes los rellenamos con el carácter 32 que en ASCII representa un espacio en blanco.

6.1.3 CONVERSION A FORMATO BINARIO.

Finalmente las cadenas decimales de tamaño fijo, deben ser convertidas a un formato que pueda generar los cambios suficientes para que en el rango de entrada de una neurona sigmoidea se genere el mayor cambio posible. Recordaremos que el rango de entrada de una neurona acepta cualquier valor de los números reales, sin embargo los cambios más significativos a la salida de estas neuronas, suceden cuando el rango de entrada esta entre cero y uno.

Esta es la principal razón para elegir que los números decimales con los que ahora contamos, que van desde 0 hasta 255, sean convertidos a formato binario para que se ajusten al rango de 0 y 1 necesario para que las neuronas trabajen adecuadamente y nos permitan que la variación en los pesos de la red neuronal, debido al entrenamiento, no sean tan drásticos. Así pues, las cadenas decimales de tamaño fijo con las que contábamos como pudieran ser las siguientes:

64 37 42

37 42 33

43 28 96

28 96 41

Serán transformadas a formato binario quedando de la siguiente manera:

01000000	00100101	00101010
00100101	00101010	00100001
00101011	00011100	01100000
00011100	01100000	00101001

Es decir que el tamaño de cada cadena pasó de ser el tamaño de la ventana estática a 8 veces este tamaño, debido a que un número que va de 0 hasta 255 se puede representar en 8 bits binarios.

En este momento contamos con las cadenas de entrenamiento en el formato adecuado para servir de información de entrada para el entrenamiento de nuestra red neuronal. Ahora solo basta definir el modelo de red neuronal más adecuado para solucionar el problema en cuestión referente a la detección de Intrusos en redes de comunicación.

Cabe mencionar que el código de este pre-procesamiento de datos esta anexado en el código fuente, y muestra que para los datos con los que contamos, se ha utilizado una ventana estática de ocho letras, por lo que al representarla en binario ha generado cadenas de tamaño fijo igual a 64 bits. Estas cadenas de 64 bits serán los elementos de entrada de nuestra red, por lo que a este momento se ha definido que el número de entradas de nuestra red neuronal deberá corresponder a esta cantidad de longitud de cadena igual a sesenta y cuatro.

6.2 DISEÑO Y ENTRENAMIENTO DE LA RED NEURONAL.

Una vez que se ha definido el número de entradas de la red neuronal igual a 64 se puede definir también el número de salidas necesarias para representar las cinco clasificaciones posibles que nuestra red neuronal debe tener, tomando en cuenta la clasificación mencionada en el capítulo 2.4 sobre la cual se debe entrenar a nuestra red. De esta manera determinamos que el número de neuronas de salida sea igual a cinco, tomando en cuenta que cada neurona sea destinada a activarse cuando el dato en la entrada de la red neuronal corresponda a un tipo de clasificación, y que las demás neuronas se encuentren desactivadas cuando el dato a la entrada no pertenezca a la clasificación determinada por

la neurona.

En otras palabras, cada una de las cinco neuronas debe corresponder a una de las cinco posibles clasificaciones de manera que cuando el dato a la entrada pertenezca a una de estas clasificaciones, la neurona correspondiente a esta clasificación sea activada y las demás sean deshabilitadas.

Para esto, se ha diseñado en el código *IDS*2, del código fuente, que la primera neurona de salida se active cuando el dato a la entrada sea de tipo NORMAL, la segunda neurona deberá activarse cuando el dato a la entrada sea de tipo ATAQUE POR INYECCION, la tercera neurona deberá activarse cuando el dato a la entrada sea de tipo ATAQUE POR MODIFICACION DE PATH, la cuarta neurona debe ser activada cuando el dato a la entrada sea de tipo ATAQUE POR INYECCION SQL, y la quinta neurona debe ser activada para cuando el dato a la entrada corresponda a un ATAQUE POR XSS (Cross Site Script). Recordando que cuando una neurona este activada o en valor alto o en uno, las demás deberán estar en valor bajo, o cero. Esto se puede apreciar claramente en la Figura 6.2.

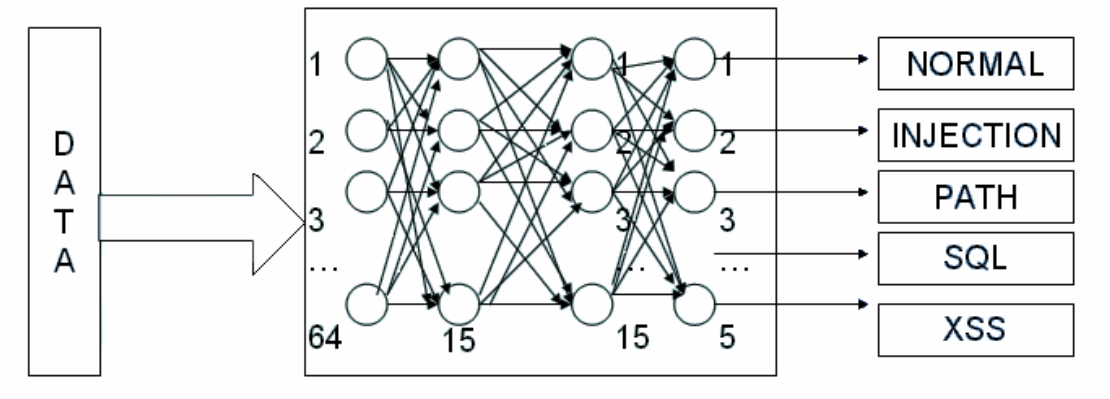

Figura 6.2 Modelo de Red Neuronal.

De esta manera, se han generado los destinos de entrenamiento, que se generan en la primera parte del código de *IDS* descrito en el Código fuente. Y con los cuales ha sido posible entrenar a la red neuronal.

Por otro lado, debe aclararse que el conjunto de datos de cadenas binarias de tamaño fijo con el que contamos ha sido dividido en dos grupos, el 70% de los datos junto con sus correspondientes destinos ha sido utilizado para entrenar a la red neuronal, mientras que el 30% restante junto con los destinos correspondientes ha sido utilizado para poner a prueba la red en la fase de prueba descrita en el siguiente apartado, de manera que la red pueda demostrar su capacidad para generalizar al probarse con datos nuevos y totalmente desconocidos.

Finalmente nos ocupa decidir el número de capas ocultas y el correspondiente número de neuronas en cada capa oculta. Siguiendo el método de diseño de prueba y error, nuestras pruebas han generado 3 modelos de redes neuronales cuya estructura cambia únicamente en el número de neuronas ocultas y en el algoritmo de entrenamiento utilizado para cada red. Definiendo la primera red como se puede apreciar en la Figura 6.2 como una red Perceptrón Multicapa de 64 entradas por 15 neuronas en la primera capa oculta y 15 más en la segunda capa oculta con las 5 neuronas de salida necesarias para expresar la identificación del dato de entrada. El algoritmo utilizado para entrenar la primera red ha sido el algoritmo de propagación elástica el cual se ha definido en la sección 4.2.2.

Cabe mencionar que al agregar una o mas capas ocultas, la red adquiere una perspectiva global y es capaz de extraer estadísticas de mayor orden. Las neuronas ocultas y su capacidad de abstracción de patrones son valorables cuando el tamaño de la capa de entrada es largo como es nuestro caso.

La segunda red neuronal que se ha puesto a prueba ha sido una red Perceptrón Multicapa de 64 entradas por 20 neuronas en la primera capa oculta, 20 neuronas en la segunda capa oculta y cinco neuronas de salida, entrenada con el algoritmo del gradiente conjugado estudiado en la sección 4.2.3 con la actualización de Polak-Ribiere. Esta red sugiere el uso de mayor memoria que la primera, debido a que el número de neuronas se ha incrementado. La red utilizada se puede ver en la siguiente figura:

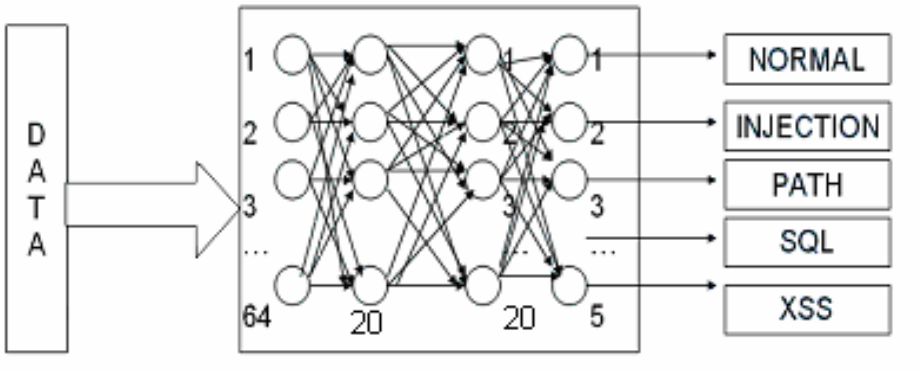

Figura 6.3 Modelo de la segunda Red Neuronal.

Finalmente la tercera red puesta a prueba ha sido la red Elman de 64 entradas por 30 neuronas en la primera capa oculta, 30 neuronas en la segunda capa oculta y cinco neuronas de salida, entrenada con el algoritmo de Propagación Elástica. El número de neuronas es considerablemente mayor que en las anteriores estructuras, debido a que las redes Elman por lo general usan más neuronas en sus capas ocultas que su semejante la red Perceptrón Multicapa para atacar un problema en particular. La arquitectura de esta red se puede observar en la siguiente figura:

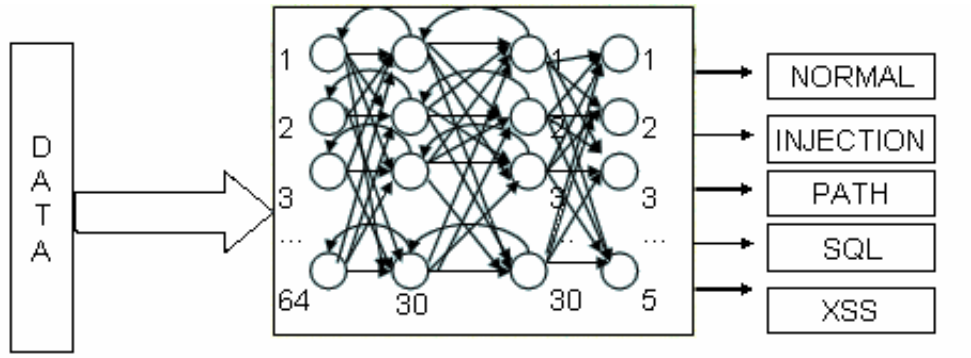

Figura 6.4 Modelo de la tercer Red Neuronal.

El entrenamiento se llevo a cabo por medio de MATLAB® a través del código presentado en el código fuente. La gráfica de entrenamiento para los datos de esta etapa con cada red neuronal se muestra en las siguientes figuras:

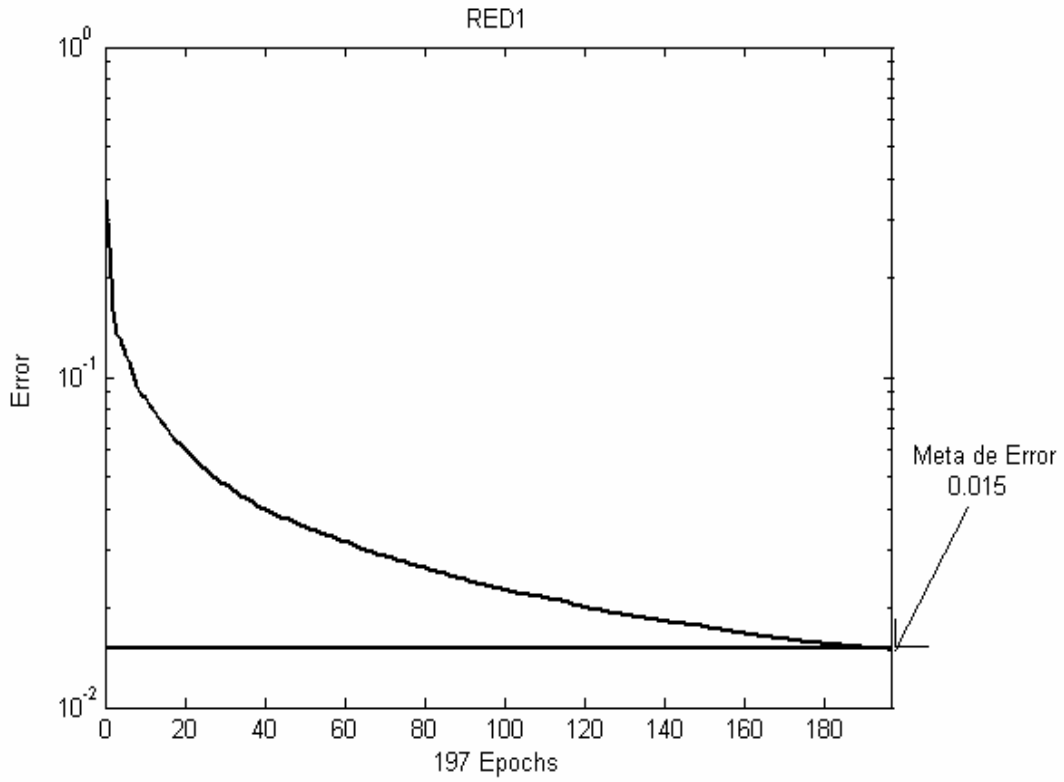

Figura 6.5 Entrenamiento de la primer Red 64X15X15X5 con Propagación Elástica.

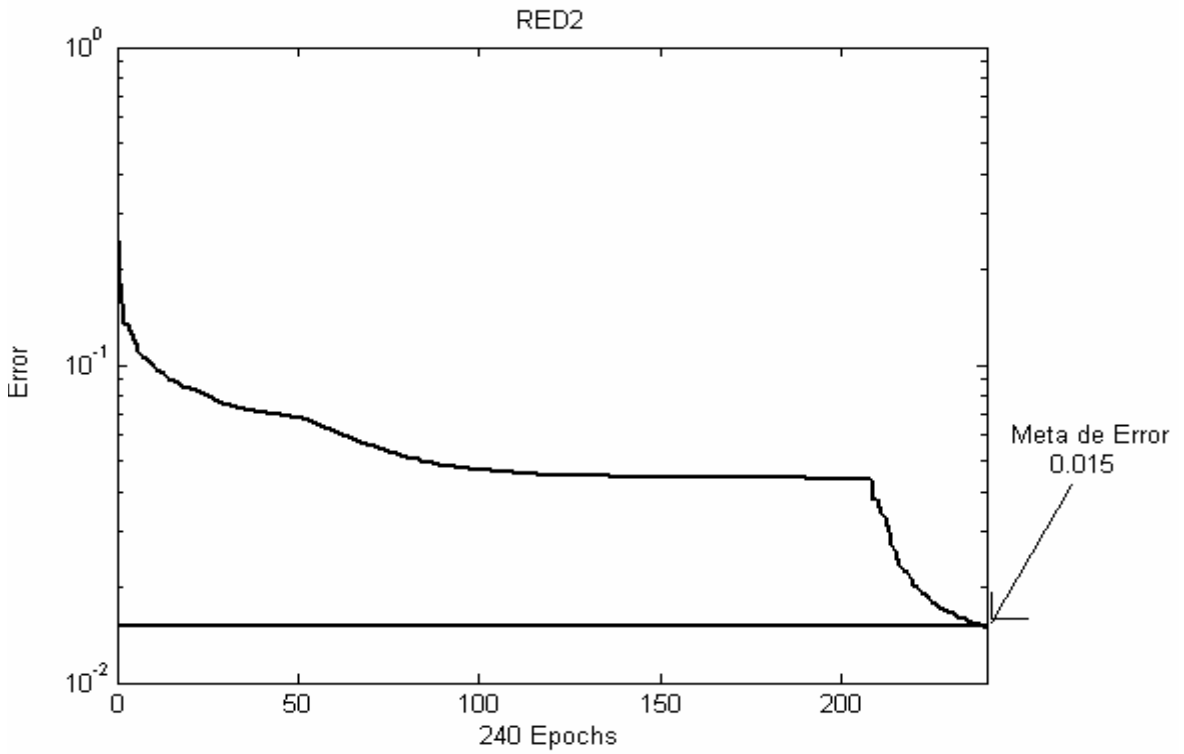

Figura 6.6 Entrenamiento de la segunda Red 64X20X20X5 con Gradiente Conjugado.

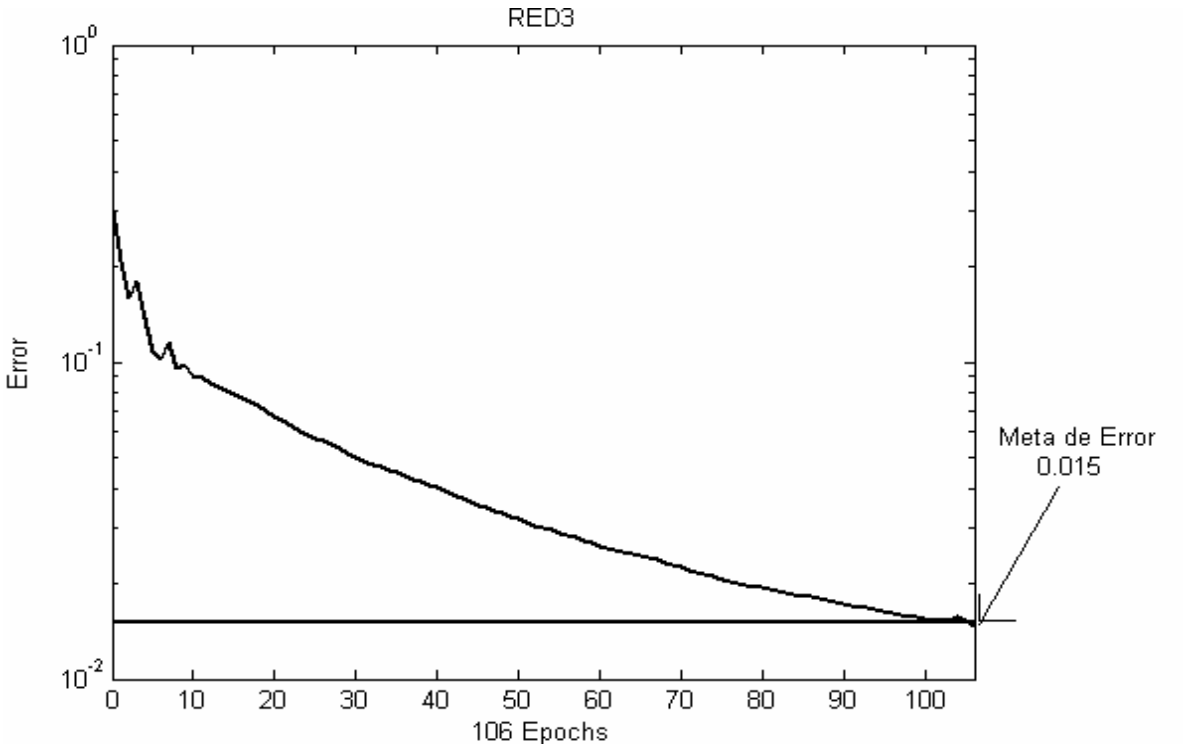

Figura 6.7 Entrenamiento de la tercer Red 64X30X30X5 con Propagación Elástica.

Se observa que rápidamente el algoritmo converge hacia un error mínimo destino, el cual, se coloco en .015. Este error no muy bajo, previene al algoritmo de sobreentrenar a la red y atenuar la generalización. El algoritmo de gradiente conjugado presentó el mayor tiempo de entrenamiento debido a los requerimientos computacionales de este tipo de algoritmo. Por otro lado la propagación elástica presentó un rápido entrenamiento debido a que esta diseñada para trabajar adecuadamente en problemas de clasificación de patrones.