Capítulo 5. Diseño del Sistema de detección de Intrusos

En este capítulo se presenta la etapa de diseño del IDS, esta etapa comprende el mecanismo de pre procesamiento de los datos que sirven como entradas al IDS y la definición de las topologías de cada uno de los modelos de redes neuronales recurrentes (la estructura de la red neuronal, sus conexiones, número de unidades y capas que la componen).

5.1. Pre procesamiento de los datos

El pre procesamiento de los datos o preparación de los datos es la etapa en la que se aplica un formato especial a los datos de entrada, en este caso tráfico de red correspondiente a requerimientos HTTP, para ser usados en el entrenamiento y prueba por la red neuronal. La estandarización de todas las variables de entrada al sistema se lleva a cabo mediante la transformación de los requerimientos HTTP a un formato binario, para llegar a este punto es necesario describir la estructura de un requerimiento HTTP.

En [SAN05] se consideran 5 categorías para clasificar los requerimientos HTTP:

1. Normal: esta clasificación define el tráfico no nocivo, es decir que no se está llevando a cabo un ataque.
2. Inyección de comando: esta categoría describe el tipo de ataque que involucra comandos ejecutados directamente en el sistema debido a vulnerabilidades en la validación de los datos, es decir, la ejecución de códigos programados en Shell.
3. Ataque *SQL*: considerado un tipo de inyección de comando, pero ejecutado en bases de datos SQL.
4. Ataque *XSS* (*Cross Site Scripting*): este tipo de ataque permite ejecutar scripts en páginas web a través de HTML, JavaScript o VBScript.
5. Modificación Ruta: Se define como la modificación de la ruta de un archivo o directorio para dar privilegios al atacante.

La estructura básica de un requerimiento HTTP es: //nombre.exe?param1=\..\archivo, como se observa, la cadena consiste en nombres de archivo, parámetros y secuencias alfa numéricas, las cuales varían de equipo en equipo. Las partes más importantes de la estructura del requerimiento HTTP son las extensiones de archivo y los caracteres especiales. El primer paso para el procesamiento de los datos es reemplazar cada cadena alfanumérica con el carácter "@", de esta manera el requerimiento anterior se transforma en: //@.exe?@=\..\@. Una vez que se ha obtenido la parte significativa de cada

requerimiento, el siguiente paso es convertir la cadena de caracteres a formato ASCII[1] decimal correspondiente a cada caracter. La Tabla 5.1 muestra ejemplos de requerimientos HTTP de cada una de las categorías junto con su conversión a valores ASCII.

Tabla 5. 1: Ejemplos de requerimientos con ventana deslizante

Requerimiento HTTP	Valor ASCII con ventana deslizante	Categoría
&@=/@/&@=@	38 64 61 47 64 47 38 64 64 61 47 64 47 38 64 61 61 47 64 47 38 64 61 64	Normal
&@=";@";	38 64 61 34 59 64 34 59	Inyección de comando
%@..%@..%@	37 64 46 46 37 64 46 46 64 46 46 37 64 46 46 37 46 46 37 64 46 46 37 64	Modificación Ruta
'--@	0 0 0 0 39 45 45 64	Ataque SQL
")</@>.jsp	34 41 60 47 64 62 46 106 41 60 47 64 62 46 106 115 60 47 64 62 46 106 115 112	Ataque XSS (*Cross Site Scripting*)

Dado que los requerimientos no tienen una longitud fija, es necesario establecer un mecanismo mediante el cual podamos fijar la longitud de las cadenas de entrada. El mecanismo que logra este objetivo se conoce con el nombre de ventana deslizante (*sliding window*) y su objetivo es convertir un vector de entradas de longitud variable en varios vectores de entrada de longitud fija. Para llevar a cabo el proceso de ventana deslizante consideramos la conversión a valores ASCII de un requerimiento http como se muestra a continuación:

38	**64**	**61**	**47**	**64**	**47**	**38**	**64**	**61**	**64**

Considerando C el tamaño fijo de los vectores de entrada, N la longitud inicial del vector a procesar, entonces en el proceso de ventana deslizante se obtendrán $N - C + 1$ vectores de longitud fija igual a C. En nuestra estructura se consideran vectores de

[1] "American Standard Code for Information Interchange" (Código Standard Norteamericano para Intercambio de Información). Este código fue propuesto por Robert W. Bemer, buscando crear códigos para caracteres alfa-numéricos (letras, símbolos, números y acentos). De esta forma sería posible que las computadoras de diferentes fabricantes lograran entender los mismos códigos.

longitud fija de 8, por lo que el vector anterior después del proceso de ventana deslizante se convierte en:

38	**64**	**61**	**47**	**64**	**47**	**38**	**64**
64	**61**	**47**	**64**	**47**	**38**	**64**	**61**
61	**47**	**64**	**47**	**38**	**64**	**61**	**64**

El paso final del pre procesado consiste en convertir las cadenas de valores decimales en su equivalente binario para alimentar las unidades de entrada de la red neuronal. Tomando en cuenta que valores ASCII pueden ser representados mediante de 8 bits, convirtiendo cada uno de los valores de los vectores a formato binario resultarán en cadenas de 64 elementos, como se muestra a continuación:

Tabla 5. 2: Conversión binaria de requerimientos

38 64 61 47 64 47 38 64	0010011001000000001111010010111101000000001011110010011001000000
64 61 47 64 47 38 64 61	0100000000111101001011110100000000101111001001100100000000111101
61 47 64 47 38 64 61 64	0011110100101111010000000010111100100110010000000011110101000000

5.2. Diseño de la Topología del IDS

Debido a que nuestros vectores de entrada tienen una longitud de 64 bits, se necesita que la capa de entrada de los modelos a probar tenga 64 unidades. Para fines de comparación se consideran cuatro arquitecturas, la primera es la recurrente totalmente conectada con función de activación sigmoide logarítmica cuyo algoritmo de aprendizaje es RTRL, la segunda es una red recurrente Elman de tres capas con función de activación tangente hiperbólica sigmoide en sus unidades de procesamiento y entrenada con el algoritmo de retro propagación, la tercera es la red neuronal wavelet con conexiones auto recurrentes en sus unidades ocultas que utiliza un algoritmo de gradiente descendente como entrenamiento y la última arquitectura representa el modelo propuesto (SRWNN – MRW) entrenado con el algoritmo deducido en la sección 4.3.

5.2.1. Arquitectura recurrente totalmente conectada

Para la arquitectura totalmente conectada se utilizaron 65 líneas de entrada, con 64 líneas de estímulos externos y una de entrada constante *bias*. Como se señala en [WIL89] los requerimientos computacionales del algoritmo RTRL son altos y se propone utilizar un número moderado de unidades de procesamiento, es decir de 20 a 30 unidades. Además de

detectar si el patrón de entrada es de naturaleza intrusiva o no, necesitamos clasificar a que categoría pertenece por lo que la arquitectura de la red habilita 5 salidas que pertenecen a cada una de las clasificaciones mostradas en la Tabla 5.3.

La operación de RTRL necesita definir la estructura Z, resultante de la concatenación de las salidas de las unidades de procesamiento (30) y las líneas de entrada (65). Por lo tanto los valores del vector Z quedan dados por:

$$Z = \begin{bmatrix} \boldsymbol{y}(t) \\ \boldsymbol{x}(t) \end{bmatrix} \tag{5. 1}$$

$$z_k = \begin{cases} x_k(t) si\ k \in I \\ y_k(t) si\ k\ \in U \end{cases}$$

$$I = \{1 \dots 65\}$$

$$U = \{1 \dots 30\}$$

donde la entrada de red a cada unidad de procesamiento está dada por:

$$s_k(t) = \sum_{l \epsilon U \cup I} w_{kl} z_l(t) \tag{5. 2}$$

$$k = 1 \dots 30$$

y la salida de cada unidad de procesamiento está dada por:

$$y_k(t+1) = f_k[s_k(t)] \tag{5. 3}$$

$$k = 1 \dots 30$$

Tabla 5. 3: Clasificación de las salidas IDS

Normal
Inyección
Path
Sql
Xss

La función de activación f_k que se considera en primera instancia es la función sigmoide logarítmica, los valores de entrada que puede tomar esta función oscilan entre

menos y más infinito, pero restringe la salida a valores entre cero y uno. Esta descrita por la fórmula:

$$f(n) = \frac{1}{1+e^{-n}} \quad (5.\ 4)$$

La Figura 5.1 muestra el comportamiento de la función sigmoide logarítmica.

La segunda función de activación f_k es la función tangente sigmoidal hiperbólica cuyo dominio es de menos infinito a más infinito y produce salidas en el rango de menos uno y más uno. La función Tangente Sigmoidal Hiperbólica es descrita mediante la siguiente fórmula:

$$f(n) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}} \quad (5.\ 5)$$

La Figura 5.2 muestra su comportamiento.

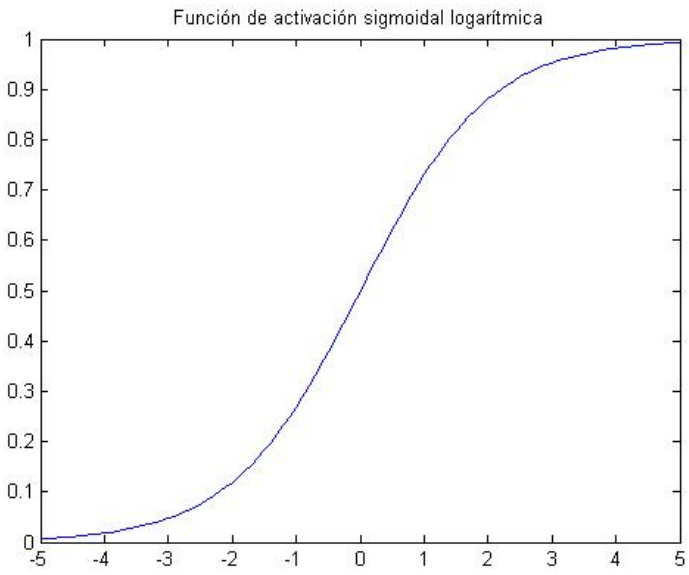

Figura 5. 1: Función sigmoide logarítmica

El algoritmo RTRL calcula la sensibilidad de las salidas de las unidades de procesamiento respecto a todos los pesos sinápticos del modelo. Dado que en el diseño del IDS se tiene 30 unidades de procesamiento con 65 líneas de entrada, la matriz de pesos sinápticos W quedaría expresada como se indica en la Figura 5.3.

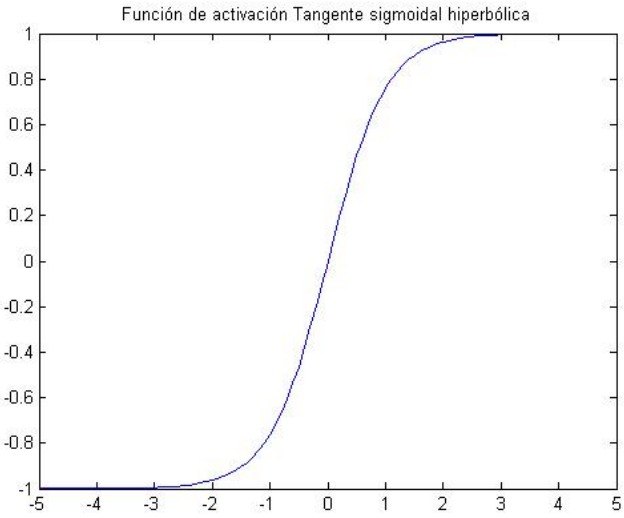

Figura 5. 2: Función tangente hiperbólica

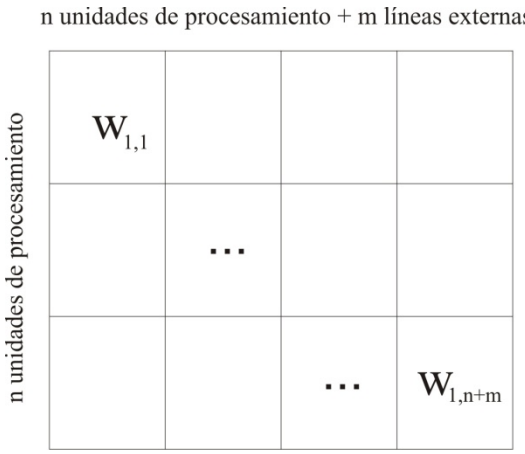

Figura 5. 3: Matriz de pesos RTRL

El proceso de hallar los valores $\frac{\partial y_k(t)}{\partial w_{ij}}$ para cada una de las salidas implica que tengamos que construir estructuras que almacenen estos valores para finalmente actualizar la matriz de pesos en cada iteración del proceso. De hecho, esta estructura define la característica de no localidad de RTRL debido a que toda la matriz de pesos y el vector de errores deben estar disponibles al momento del cálculo de las derivadas parciales.

5.2.2. Red Elman

La red Elman contempla un vector de 64 entradas en la capa de entrada, 30 unidades en cada capa oculta y 5 unidades de salida. En el caso de las unidades en capas ocultas, se opto

por la función de transferencia tangente hiperbólica (ver Figura 5.2). Para las unidades de salida se seleccionó como función de transferencia a la función sigmoide logarítmica (ver Figura 5.1).

5.2.3. SRWNN

De igual forma que la arquitectura anterior, la red wavelet con conexiones auto recurrentes tiene una capa de entrada con 64 unidades, las cuales corresponden al formato descrito en la sección anterior. En la capa *wavelon* tenemos 256 unidades de procesamiento mientras que en la capa de producto 4 unidades que multiplican las salidas de las unidades de la capa anterior y finalmente en la capa de salida 5 unidades. El criterio para determinar el número de unidades de procesamiento en la capa wavelet está relacionada con la inicialización de los *wavelons*, dado que para cada entrada se ha definido la división del dominio de la señal de entrada en potencias de dos [ZHA92].

Cada entrada al modelo es procesada por cuatro unidades en la capa *wavelon*, cada una de estas unidades de procesamiento utiliza una función wavelet como función de transferencia. La función wavelet madre seleccionada fue la primera derivada de la función Gaussiana. Esta función wavelet se muestra en la Figura 5.4 y tiene la siguiente fórmula:

$$\psi(x) = -xe^{-\frac{1}{2}x^2} \tag{5. 6}$$

La función de transferencia wavelet de cada unidad *wavelon* ψ_{jk} es obtenida a partir de la función wavelet madre por medio de los parámetros de traslación m_{jk} y escalamiento d_{jk} como sigue:

$$\psi_{jk}(z_{jk}) = \psi\left(\frac{u_{jk} - m_{jk}}{d_{jk}}\right) \tag{5. 7}$$

$$z_{jk} = \frac{u_{jk} - m_{jk}}{d_{jk}}$$

$$j = 1 \ldots 4$$

$$k = 1 \ldots 64$$

La inicialización de los parámetros de traslación y escalamiento se lleva cabo mediante una función recurrente en la que se busca cubrir todo el dominio de la función modificando la función wavelet madre [ZHA92].

La entrada u_{jk} a cada nodo wavelet esta afectada por el peso sináptico recursivo θ_{jk} de ese nodo y el valor del nodo wavelet en el tiempo anterior $\psi_{jk}(n-1)$, la entrada queda establecida:

$$U_{jk}(n) = x_k(n) + \psi_{jk}(n-1)\theta_{jk} \tag{5. 8}$$

$$j = 1 \ldots 4$$

$$k = 1 \ldots 64$$

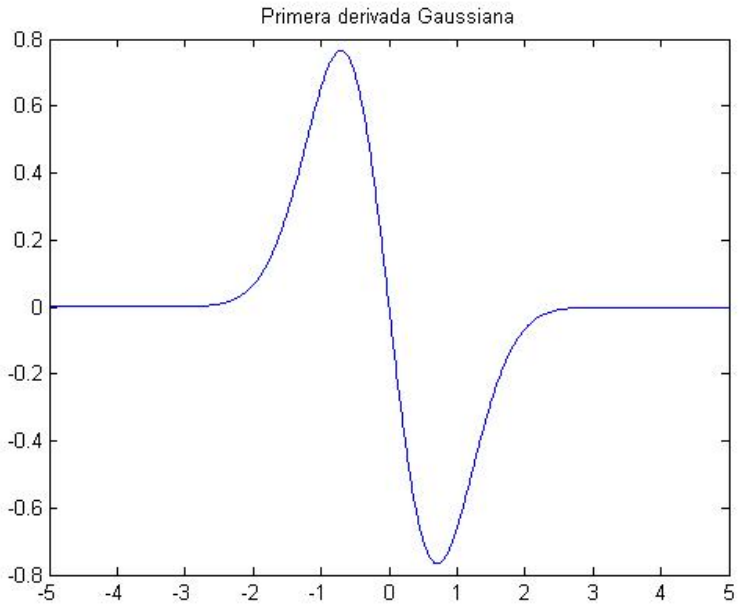

Figura 5. 4: Función wavelet primera derivada Gaussiana

La capa de multiplicación lleva a cabo la multiplicación de las salidas de la capa anterior, por lo que para cada una de las 4 unidades su salida está dada por:

$$\Phi_j(\boldsymbol{x}) = \prod_{k=1}^{64} \psi(z_{jk}) \tag{5. 9}$$

$$j = 1 \ldots 4$$

Finalmente cada una de las cinco salidas formada por un combinador lineal queda expresada por la siguiente ecuación:

$$y(n) = \sum_{j=1}^{4} w_j \Phi_j(\boldsymbol{x}) + \sum_{K=1}^{64} a_k x_k \tag{5. 10}$$

En la Figura 5.5 se muestra la arquitectura de la red wavelet con conexiones auto recurrentes, formada por cuatro capas: la capa de entrada, capa *wavelon*, capa multiplicación y capa de salida.

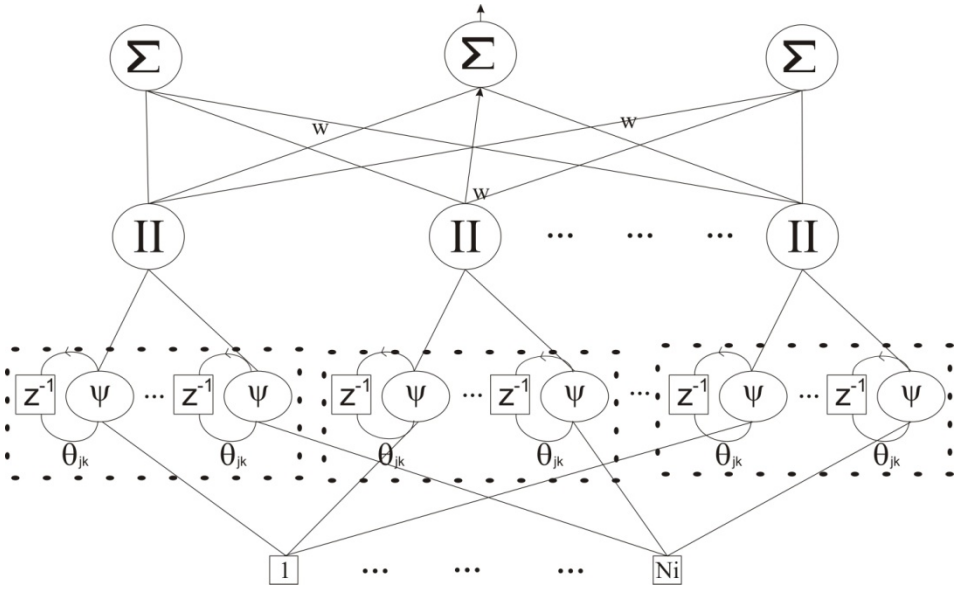

Figura 5. 5: Arquitectura SRWNN

5.2.4. SRWNN – MRW

Como todas las arquitecturas anteriores, el número de entradas se establece a 64, haciendo el valor de $N_i = 64$. En la capa multidimensional radial se estableció el número de 4 unidades $N_{MRW} = 4$, de esta manera se tienen 4 unidades R y 4 unidades ψ. La elección del número de unidades $N_{MRW} = 4$ obedece a la definición del *grid* diádico establecido en [ZHA92]. Finalmente el número de combinadores lineales en la capa de salida se establece en 5, haciendo $N_s = 5$.

El vector de entrada se define por:

$$\boldsymbol{x}(n) = [x_1 \dots x_{64}] \tag{5. 11}$$

las 64 entradas se propagarán a través de los 4 nodos R, para procesar las 64 entradas dentro de los nodos es necesario establecer los parámetros de traslación y escalamiento de la siguiente manera:

$$\boldsymbol{d}_j = [d_{j,1} \dots d_{j,64}] \tag{5. 12}$$

$$\boldsymbol{t}_j = [t_{j,1} \dots t_{j,64}] \tag{5. 13}$$

$$j = 1 \dots 4$$

Las salidas de las unidades R se denotan por medio de a_j, y representan parte de la entrada a las unidades wavelet. El término restante de la entrada está definido por un patrón

temporal que denota el estado pasado de las unidades wavelet, siendo el resultado de la conexión recurrente definida en el modelo propuesto:

$$\alpha_j = a_j + \psi_j (n-1)\theta_j \qquad (5.14)$$

$$j = 1 \dots 4$$

de esta manera α_j representa la entrada al nodo wavelet, como función wavelet madre se escogió de la misma manera que en el modelo SRWNN a la primera derivada Gaussiana. Las salidas de las unidades wavelet está dada por $\psi(\alpha_j)$.

Las unidades de salida están constituidas por 5 combinadores lineales ($i = 1 \dots 5$), cada combinador lineal representa una salida del modelo y tiene conexiones con cada nodo wavelet de la capa anterior, dichas conexiones tienen asociado un peso w_{ij} el cual se modifica en la etapa de entrenamiento. Además de considerar los elementos de la capa anterior, las unidades de salida tienen conexiones directas con las entradas, afectadas por medio de un factor a_{ik} también modificado durante el entrenamiento. La salida del modelo se describe por medio de:

$$y^i(n) = \sum_{j=1}^{4} w_{ij}\, \psi(\alpha_j) + \sum_{k=1}^{64} a_{ik}\, x_k \qquad (5.15)$$

$$i = 1 \dots 5$$

5.2.5. Tasa de aprendizaje

La tasa de aprendizaje es el parámetro que establece que tan rápido se efectúa el proceso de aprendizaje y que tan efectivo es. Una red neuronal que se encuentra en el proceso de entrenamiento modifica sus pesos de tal manera que pueda reconocer o llevar a cabo el mapeo de entradas y salidas de todas las muestras expuestas hasta el momento, el ajuste de los parámetros de la red se lleva a cabo por medio de un mecanismo iterativo que toma en cuenta la tasa de aprendizaje para cambiar su valor en una fracción. El valor de la tasa de aprendizaje se encuentra en el intervalo de cero y uno, el escoger un valor muy cercano a cero requerirá un número alto de épocas de entrenamiento ocasionando que el proceso de aprendizaje sea muy lento. Al contrario, si el valor de la tasa de aprendizaje es muy alto o cercano a uno se tendrá un comportamiento oscilatorio que ocasione que la red se encuentre en un estado de inestabilidad [HAY98].

5.2.5.1. Tasa de aprendizaje adaptativa

Los algoritmos tradicionales de aprendizaje para redes neuronales mantienen una alta dependencia al valor inicial asignado al parámetro de tasa de aprendizaje. Otro problema que afecta a implementaciones como retro propagación simple es la existencia de mínimos

locales, estas regiones ocasionan que la minimización de la función de error se dirija a este mínimo y que se detenga cuando se alcance; cuando en realidad lo que se quiere lograr es el alcance de un mínimo global. A lo largo del tiempo se ha descubierto que la manipulación de la tasa de aprendizaje durante el proceso de entrenamiento puede producir mejores resultados y de esta manera han surgido distintos métodos para llevar a cabo este mecanismo de adaptación.

El enfoque estático para la tasa de aprendizaje considera un valor fijo a lo largo del proceso de aprendizaje, esto ocasiona que el desempeño del algoritmo sea muy sensible a los cambios que se efectúen en esta cantidad. Si la tasa de aprendizaje es establecida a un valor numérico muy alto, el algoritmo puede oscilar y volverse inestable. Por otra parte si la tasa de aprendizaje es muy pequeña el algoritmo tarda mucho tiempo en converger. En la actualidad no existe un método práctico para establecer la tasa de aprendizaje a un valor óptimo al momento de inicializar la estructura de la red neuronal, además, el valor óptimo de la tasa de aprendizaje varía durante todo el proceso de aprendizaje a medida que el algoritmo se mueve por la superficie de error.

Se puede mejorar el desempeño en el proceso de entrenamiento si se permite que la tasa de aprendizaje varíe según se necesite durante la etapa de aprendizaje. El mecanismo de adaptación de la tasa de aprendizaje modifica su valor para mantener el tamaño de los pasos de aprendizaje tan grandes como sea posible manteniendo siempre la estabilidad del algoritmo. Los algoritmos de aprendizaje para redes neuronales buscan la minimización de una función de error iterativamente, ajustando sus parámetros libres en cada iteración para producir un resultado que represente el desempeño del sistema. La función de error a ser minimizada a menudo es expresada como una suma promedio de las diferencias entre los valores deseados y obtenidos para cada una de las unidades de procesamiento. Algunos autores utilizan el promedio del error cuadrático de tal manera que el error obtenido no dependa del tamaño del tamaño del conjunto de patrones de entrenamiento ni del número de unidades de salida, en este caso la función del error está dada por la siguiente fórmula [MOR95]:

$$E = \frac{1}{N_l P} \sum_{p=1}^{P} \sum_{j=1}^{Nl} (des_j - obt_j)^2 \qquad (5.16)$$

donde N_l representa el número de unidades de salida y P es el número de patrones de entrenamiento disponibles para una época.

Durante la etapa de entrenamiento, un conjunto de patrones son usados, dichos patrones tienen un valor objetivo para cada unidad de salida. Cada patrón de entrenamiento al ser presentado a la capa de entrada propaga su señal a través de las distintas conexiones de la arquitectura de la red neuronal para producir un conjunto de salidas que se utilizan para calcular el error en esa iteración. Las correcciones a los parámetros libres de la red son ajustadas dependiendo del valor de error encontrado en cada salida, buscando adaptar el

comportamiento de la red a los valores deseados. Este proceso iterativo se repite hasta que todos los patrones se han presentado a la red, tras lo cual se dice que se ha llevado a cabo una época de entrenamiento [HAY98].

La tasa de aprendizaje adaptativa se modifica en cada época, de esta manera los ajustes a los parámetros de la red usan en cada época un valor distinto de la tasa de aprendizaje. El valor nuevo de la tasa de aprendizaje debe validarse para encontrar el valor máximo que asegure la convergencia del algoritmo sin que se vuelva inestable. El aumento en el valor de la tasa de aprendizaje en cada época es comparado con el aumento en la función de error, en caso de que el nuevo error calculado exceda el error en la época anterior por un factor establecido los nuevos valores de los parámetros ajustables son descartados y el valor de la tasa de error es disminuido por un factor. Si por otra parte el valor de error calculado es menor, entonces los nuevos valores de los parámetros son establecidos y el valor de la tasa de aprendizaje es incrementado por un factor. Este procedimiento aumenta la tasa de aprendizaje hasta el punto en el la red neuronal siga aprendiendo sin incrementar notablemente el error.

La minimización de la función de error se hace mediante el método de gradiente descendente, en el cual los ajustes a los parámetros de la red se obtienen al calcular la derivada parcial de la función de error en relación con el parámetro que se quiere ajustar. Dado que se busca un proceso de minimización, los valores de ajuste de cada parámetro usan el valor negativo del vector de gradiente. La regla delta determina la cantidad de ajuste de cada parámetro basándose en la dirección del gradiente y en el valor de la tasa de aprendizaje [YOO05]:

$$W(n+1) = W(n) + \Delta W(n) \tag{5.17}$$

$$\Delta W(n) = -\eta \frac{\partial E}{\partial W(n)}$$

donde El parámetro η representa el tamaño del paso o es llamado también la tasa de aprendizaje.

Existen dos variantes en la presentación del conjunto de patrones de aprendizaje a la red neuronal. Con el aprendizaje en línea cada patrón es presentado a la red neuronal, la señal de entrada se propaga por toda la estructura y se obtienen las salidas del sistema, a continuación la regla delta es usada para calcular los valores de los gradientes y llevar a cabo la actualización de los parámetros. Con el aprendizaje en lote o “batch”, los parámetros son actualizados después de cada época usando la suma de los gradientes calculados en cada iteración.

El valor de la tasa de aprendizaje debe adaptarse para mantener un número lo suficientemente grande para permitir un proceso de aprendizaje rápido y al mismo tiempo

ser suficientemente pequeño para mantener la efectividad. Es decir que si la búsqueda del mínimo global nos lleva a una pendiente pronunciada, se intenta mantener la tasa de aprendizaje lo suficientemente pequeña para alcanzarlo que de otra manera solamente oscilaría entre los limites sin llegar a encontrarlo. Por otra parte si la búsqueda del mínimo global nos lleva a un espacio regular con tendencia continua la tasa de aprendizaje debe ajustarse para moverse más rápido en la dirección descendente.

En [MOR95] se presenta una taxonomía completa de técnicas de optimización para la adaptación de la tasa de aprendizaje, dividida de la siguiente manera:

- Procedimiento basados en optimización numérica
  - Gradiente conjugado
  - Quasi – Newton
  - Cálculos de segundo orden para tasa de aprendizaje
- Optimización estocástica
- Basados en heurísticas

Para el caso de la arquitectura auto recurrente con wavelets, en [YOO05] se establecen los teoremas y demostraciones que permiten conocer los valores máximos de las tasas de aprendizaje para cada uno de los parámetros variables (ver Tabla 5.4).

Tabla 5. 4: Tasas de aprendizaje para cada parámetros

η_I^a	Tasa de aprendizaje para los pesos que conectan la capa de entrada con los combinadores de la capa de salida.
η_I^m	Tasa de aprendizaje para los parámetros de traslación de la wavelet.
η_I^d	Tasa de aprendizaje para los parámetros de escalamiento de la wavelet.
η_I^θ	Tasa de aprendizaje para los pesos recursivos de las unidades wavelet.
η_I^w	Tasa de aprendizaje de los pesos que se encuentran entre la capa de producto y la capa de salida.

Cada una de las tasas de aprendizaje toma como valor máximo que garantiza la convergencia asintótica, las siguientes cantidades [YOO05]:

Tabla 5. 5: Valores máximos permitidos para tasas de aprendizaje

$\boldsymbol{\eta_I^a}$	$\boldsymbol{\eta_I^{a,M} = \frac{1}{N_i \left\| x_{i,max} \right\|^2}}$
$\boldsymbol{\eta_I^m}$	$\eta_I^{m,M} = \frac{1}{N_w N_i} \left[\frac{1}{\left\| w_{i,max} \right\| \left(\frac{2\exp(-0.5)}{\left\| d_{i,min} \right\|} \right)} \right]^2$
$\boldsymbol{\eta_I^d}$	$\eta_I^{d,M} = \frac{1}{N_w N_i} \left[\frac{1}{\left\| w_{i,max} \right\| \left(\frac{2\exp(-0.5)}{\left\| d_{i,min} \right\|} \right)} \right]^2$
$\boldsymbol{\eta_I^\theta}$	$\eta_I^{m,M} = \frac{1}{N_w N_i} \left[\frac{1}{\left\| w_{i,max} \right\| \left(\frac{2\exp(0.5)}{\left\| d_{i,min} \right\|} \right)} \right]^2$
$\boldsymbol{\eta_I^w}$	$\eta_I^{w,M} = \frac{1}{N_w}$

donde:

Tabla 5. 6: Notación utilizada

$\boldsymbol{N_i}$	Número de entradas de la red neuronal auto recurrente.
$\boldsymbol{x_{i,max}}$	Denota el máximo valor de los valores absolutos de todo el conjunto de entradas.
$\boldsymbol{N_w}$	Número de unidades wavelet asignadas a cada entrada.
$\boldsymbol{w_{i,max}}$	Denota el máximo valor de los valores absolutos de los pesos de salida.
$\boldsymbol{d_{i,min}}$	Denota el mínimo valor de los valores absolutos de los factores de escalamiento.

5.3. Discusión

Como parte del proceso de preparación de los datos que sirven como conjunto de entrenamiento para las redes neuronales, es necesario pre procesar las entradas a un formato adecuado antes de presentar los datos a la capa de entrada de las distintas topologías. Además, en este capítulo se establecieron las topologías de las redes a comparar en la implementación del IDS.

Tomando en cuenta las necesidades para la aplicación del IDS, para cada topología se presentó el número de unidades en la capa entrada, número de unidades en las capas

intermedias, capa de salida así como las interconexiones entre las unidades. Se conocieron las características de las estructuras de datos que son necesarias para almacenar los parámetros ajustables y la manera en la que se llevará a cabo la modificación de estos.

En la parte final de este capítulo se presentó la importancia de la tasa de aprendizaje en el proceso de entrenamiento y la manera en que la adaptación de este parámetro es utilizado para mejorar la velocidad de convergencia de los algoritmos.