CAPITULO 3 PERCEPTRON MULTICAPA.

En el este capítulo se introducen los conceptos que describen a las redes neuronales como tecnología adaptiva de reconocimiento de patrones, analizando su estructura, algoritmo de entrenamiento y presentando su uso para clasificación de patrones.

3.1 EL PERCEPTRÓN Y SUS LIMITACIONES.

3.1.1 DEFINICION DE NEURONA ARTIFICIAL

Cuando hablamos de sistemas neuronales es importante definir el concepto de neurona artificial, elemento básico de este tipo de sistemas, cuya descripción se hereda de los principios neuro-biológicos que describen el comportamiento de las neuronas en la corteza cerebral. La neurona artificial consta de elementos de entrada y salida que se procesan en la unidad central, así como de los elementos de procesamiento que permitirán a la neurona generalizar y aprender conceptos. La arquitectura básica de la neurona se muestra a continuación [HAY94].

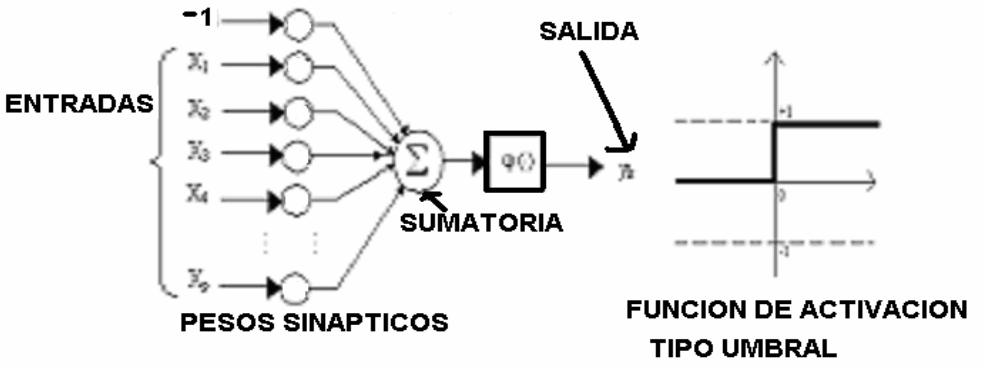


Figura 3.1 Arquitectura de la neurona y función de Activación.

Donde $X_{0\ldots P}$ es el vector de entrada de la neurona, $\omega_{k0\ldots kP}$ es el vector de pesos correspondientes a la neurona k por los que se multiplica cada entrada, a θ_k lo llamaremos el umbral que se multiplica por una entrada $X_0 = -1$, υk es el producto punto del vector de

entrada X_k y el vector de pesos ω_k, φ () es la función de activación y finalmente y_k es la salida de la neurona k. La Figura 3.1 también muestra la función de activación $\varphi()$ correspondiente a la neurona y que en este caso es la función de umbral la cual propone que a todos los valores de υk menores a cero les corresponda una salida de 0 y a todos los valores de υk mayores o iguales a cero les corresponda una salida de +1.

A partir de esta estructura básica la neurona puede mapear las entradas para obtener a la salida, una respuesta deseada que pudiera pertenecer a alguna función determinada y que, debido a la función de activación φ () que la genera, puede caer en dos categorías. Esta respuesta depende en primer lugar de las entradas de la neurona y en segundo lugar de las operaciones que se realicen al interior de la neurona.

El objeto de este esquema es plantear la posibilidad de que las operaciones internas, que están determinadas por los parámetros libres de la red, como es el vector de pesos ω y el umbral $\theta=\omega_0$, puedan representar una variedad de salidas con solo modificar dichos parámetros. De manera tal que cuando pretendamos entrenar a la neurona, para que con unas entradas nos entregue determinadas salidas, únicamente sea necesario modificar el valor de los parámetros libres para que la diferencia entre el valor esperado y el valor obtenido a la salida sea mínima. Este procedimiento básico sienta las bases de entrenamiento para los sistemas basados en la neurona como unidad básica [HAY94].

3.1.2 PROCESOS DE APRENDIZAJE EN SISTEMAS NEURONALES.

En general cualquier sistema neuronal puede utilizar distintos paradigmas para el aprendizaje de la red, al igual que distintos algoritmos de entrenamiento. La Figura 3.2 muestra el proceso de aprendizaje divido en los paradigmas y algoritmos anteriormente mencionados [HAY94].

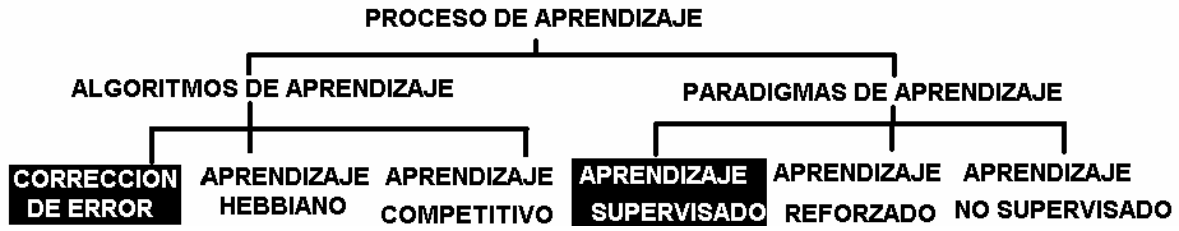


Figura 3.2. Proceso de Aprendizaje.

En las redes neuronales utilizadas en este proyecto se utiliza el paradigma de **aprendizaje supervisado** y el algoritmo de **corrección de error**, algunas veces conocido como la regla delta. Cuando hablamos de *aprendizaje supervisado (supervised learning)*, nos referimos al tipo de entrenamiento en el cual, se provee al sistema con información de las entradas al igual que se proveen las salidas esperadas o destinos correspondientes a dichas entradas a modo de que el sistema tenga los destinos como punto de referencia para evaluar su desempeño en base a la diferencia de estos valores y modificar los parámetros libres en base a esta diferencia como lo muestra la Figura 3.3.

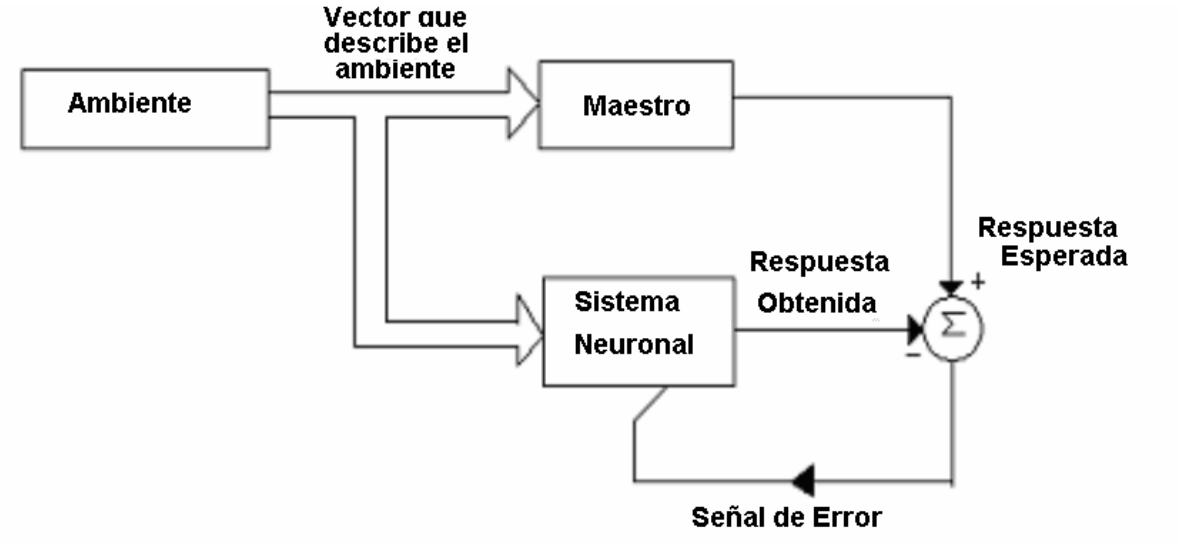


Figura 3.3. Diagrama a Bloques del Aprendizaje Supervisado.

En un aprendizaje no supervisado los parámetros libres del sistema son modificados únicamente en base a las entradas del sistema de manera que aprenden a categorizar las entradas y clasificarlas sin necesidad de una referencia. Este paradigma se plantea inadecuado para el problema de la detección de intrusos ya que no posee información para decidir si un tipo de comportamiento es normal o anómalo.

Por otro lado se puede definir una regla de aprendizaje como el procedimiento para modificar los parámetros libres del sistema neuronal para entrenar al sistema neuronal en el desempeño de una tarea. De igual forma cuando hablamos de la *regla delta* o el algoritmo de *corrección de error* nos referimos al modelo de aprendizaje que se basa en minimizar una función de error J en relación a los parámetros libres de nuestro sistema.

Una vez definida la función de costo a minimizar la corrección de error es un problema estrictamente de optimización. Para esto la regla nos indica que hay que determinar la diferencia o *delta* que debe ser agregada a cada parámetro libre para que la función de costo sea mínima [HAY94]. Básicamente si estamos hablando de que los parámetros libres en un sistema neuronal son los pesos ω y si consideramos que el umbral puede ser considerado como un peso ω_0, el algoritmo de la regla delta quedará como sigue:

- **Estimular el sistema Neuronal**. Se obtiene la señal de error e(n) a partir de las señales destino esperada d(n) y la obtenida y(n) para la muestra n.

$$e(n) = d(n) - y(n) \qquad \textbf{(3.1)}$$

- **Traspasar Cambios en el sistema Neuronal.** Una vez obtenido el error, este se minimiza hasta que el valor actual de la salida obtenida se acerque al destino esperado. Este proceso llevara a cabo cambios iterativos en el sistema neuronal de manera que a cada iteración se obtenga un error y se hagan modificaciones en los parámetros libres de la red hasta obtener un error mínimo.
- **Responder a una nueva forma de ambiente.** Una vez realizados los cambios necesarios. El nuevo ajuste es puesto a prueba nuevamente estimulando el sistema en un nuevo ambiente, observando la habilidad del sistema para generalizar.

3.1.3 EL PERCEPTRON

A partir del modelo de la neurona artificial Rosenblatt (1958,1962) desarrollo el modelo del Perceptrón el cuál básicamente consiste en una capa de neuronas con pesos y umbral ajustables como se muestra en la Figura 3.4, este sistema neuronal puede ser llamado una red neuronal debido a las conexiones existentes en sus elementos.

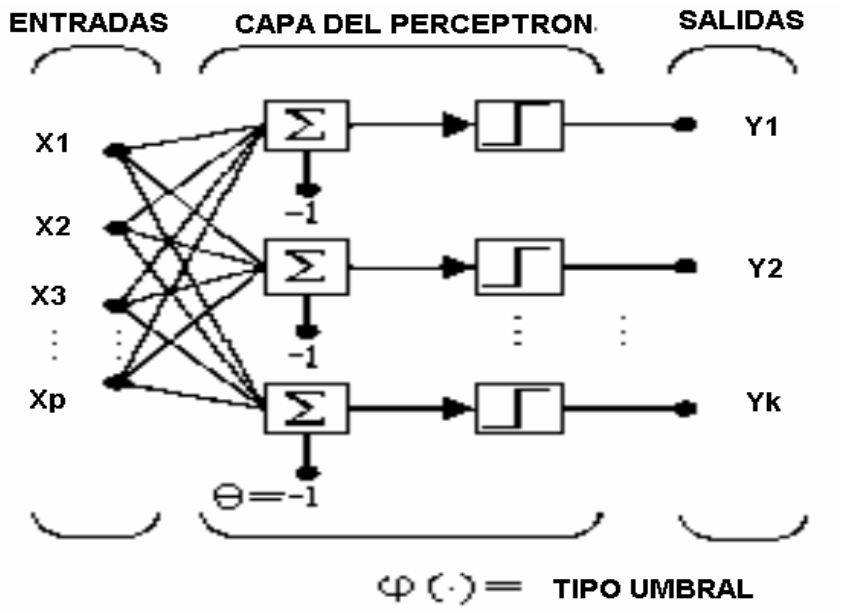


Figura 3.4. Arquitectura del Perceptrón.

Fue el mismo Rosenblatt quien en base al modelo de aprendizaje de la regla delta, planteado anteriormente, determino el algoritmo de entrenamiento del Perceptrón que, siguiendo los principios de la regla, consiste en determinar el ajuste que se debe realizar a cada peso ω en la neurona para que el error a la salida sea cero. El algoritmo de entrenamiento del Perceptrón comienza por inicializar los parámetros libres (pesos y umbrales) en cero y posteriormente se estimula la red con algún vector de entrada obteniendo el error a cada salida como:

$$e_k(n) = d_k(n) - y_k(n) \qquad \textbf{(3.2)}$$

Donde $e_k(n)$ corresponde al error a la salida para la muestra n de la neurona k, $d_k(n)$ es el valor destino esperado a la salida para la muestra n y $y_k(n)$ es la salida obtenida en la neurona k para la muestra n. El siguiente paso es calcular la delta o diferencia a agregar a los pesos para que el error e(n) sea cero. La delta esta dada por la siguiente ecuación.

$$\begin{aligned} \Delta\omega &= e(n)\,x(n) \\ \Delta\theta &= e(n) \end{aligned} \qquad \textbf{(3.3)}$$

Donde Δθ es la modificación que se debe realizar al umbral y Δω es la modificación que se debe realizar a los pesos. La regla de aprendizaje del Perceptrón puede ser expresada como sigue:

$$\begin{aligned} \omega^{nueva} &= \omega^{vieja} + e(n)\,x(n) \\ \theta^{nueva} &= \theta^{vieja} + e(n) \end{aligned} \qquad \textbf{(3.4)}$$

Si aplicamos esta regla a nuestro Perceptrón con distintas muestras, hasta que el error sea cero, se obtendrá una red que pueda generar exactamente las salidas deseadas para determinadas entradas logrando así que el Perceptrón aprenda una función.

Como se puede observar la regla delta para el Perceptrón consiste en agregar a cada peso el producto del error generado por la neurona con las entradas correspondientes de manera que se dan 3 casos posibles.

1. El error es cero y los pesos no se modifican.
2. El error es 1 y a cada peso se le agrega exactamente la entrada correspondiente.
3. El error es -1 y a cada peso se le substrae exactamente la entrada correspondiente.

3.1.4 LIMITACIONES DEL PERCEPTRON

Se debe tener en cuenta que no siempre el algoritmo de entrenamiento del Perceptrón podrá converger hacia un error nulo. De hecho el Perceptrón es incapaz de converger en aquellas funciones que no son linealmente separables, es decir, aquellas cuyos elementos pueden ser separados por una línea recta.

Esto se debe a las propiedades inherentes de las unidades básicas del Perceptrón que son las neuronas artificiales, cuya limitación reside principalmente en la función de activación que como se puede observar separa las entradas en dos salidas de manera que con todas las entradas posibles se forma un hiperplano cuyos elementos de salida son separados en dos partes dentro del hiperplano [BEA03].

Visto de otra forma, podríamos decir que el Perceptrón divide en dos grupos las entradas por medio de una línea divisoria de manera que no es posible separar elementos que no se encuentren claramente separados de otros elementos. Es decir que no se puede caracterizar elementos no lineales.

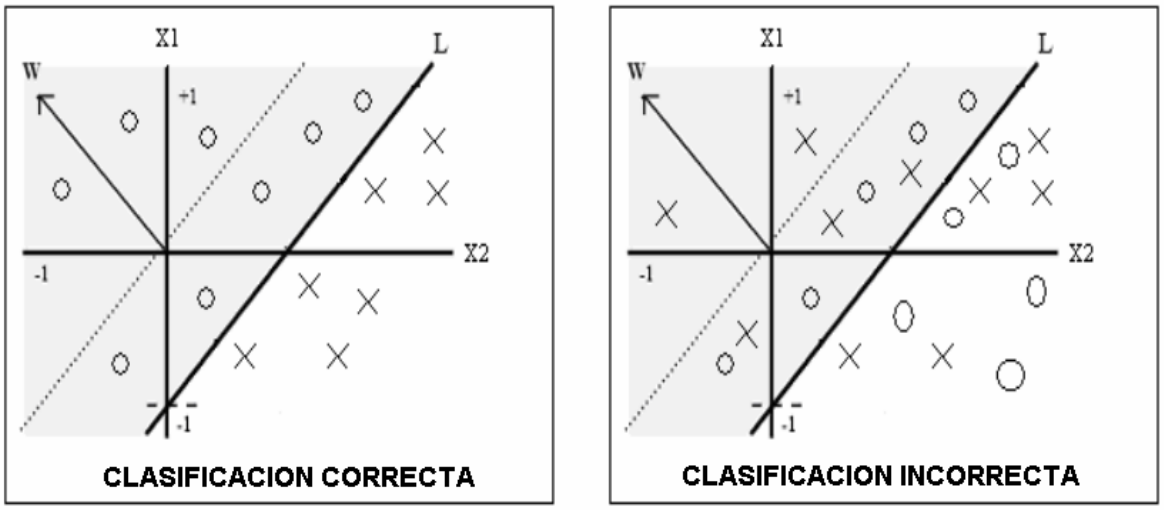


Figura 3.5. Margen de Caracterización del Perceptrón.

Esto se puede observar claramente en la Figura 3.5 donde se muestra el caso en el que el Perceptrón es un buen clasificador, y otro caso donde el Perceptrón no puede separar los elementos en dos categorías. En la figura, los puntos (X1, X2) corresponden a la intersección de dos entradas X1 y X2.

Enfocándonos al problema de la detección de Intrusos, inferimos que el uso del Perceptrón sería inadecuado debido a las limitaciones lineales que su estructura posee. Sin embargo, a partir de estos principios se comenzará a vislumbrar las características necesarias para que nuestro sistema neuronal pueda clasificar patrones no lineales como los que se presentan comúnmente en una red de comunicaciones.

En el diseño de un sistema de detección de intrusos basado en sistemas neuronales es importante conocer los principios que llevan a este tipo de sistemas a realizar clasificaciones sobre vectores de entrada ya que nos ayudará a ubicar paulatinamente el tipo de diseño que pudiera ser el más adecuado para enfrentar el problema en cuestión.

Como se podrá ir viendo más adelante, el uso del paradigma de *aprendizaje supervisado* se presenta más adecuado para solucionar el problema de la clasificación binaria de datos y los distintos algoritmos que pudieran ajustar los parámetros libres de nuestra red, son todos basados en los principios asentados en este capítulo por la *regla delta*. Únicamente cambiaran la función de costo a minimizar y la forma de minimizarla, sin embargo, en principio, el procedimiento será el mismo.

3.2 EVOLUCIÓN AL PERCEPTRÓN MULTICAPA.

3.2.1 INTRODUCCION

Como se mencionó anteriormente los modelos que se desprenden del Perceptrón se basan en los principios de corrección de error planteados por el algoritmo de la regla delta para entrenar a estos sistemas. En un principio el desarrollo del Perceptrón llevó a la generación de un nuevo tipo de red cuya modificación principal respecto a la estructura del Perceptrón se basa en el uso de varias capas de neuronas artificiales, en vez de usar una sola capa. Este hecho significativo no hubiera servido de nada sin el cambio de la función de activación de las neuronas artificiales pasando de una función no diferenciable como era la activación logística a una función diferenciable y no lineal como lo es la sigmoide que se muestra en la Figura 3.6.

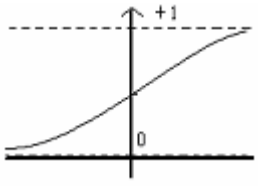


Figura 3.6. Función de Activación Sigmoide.

El recurso de este tipo de función de activación introdujo un nuevo paradigma en el procesamiento de los sistemas neuronales permitiendo a las redes neuronales aprender las variaciones no lineales de los distintos tipos de ambientes, que en su mayoría, presentan variaciones del tipo no lineal.

En este momento se puede entender la importancia de este suceso ya que la mayor parte del tiempo el flujo de datos en las redes de comunicaciones sucede aleatorio y discontinuo. Este tipo de características son precisamente las que el nuevo sistema neuronal nos permitirá asimilar. Este sistema neuronal considerado también una red neuronal se conoce en la literatura como Perceptrón Multicapa debido a que parte del principio del Perceptrón simple. A continuación se muestra la arquitectura de esta red neuronal que se analizará en el resto del capítulo.

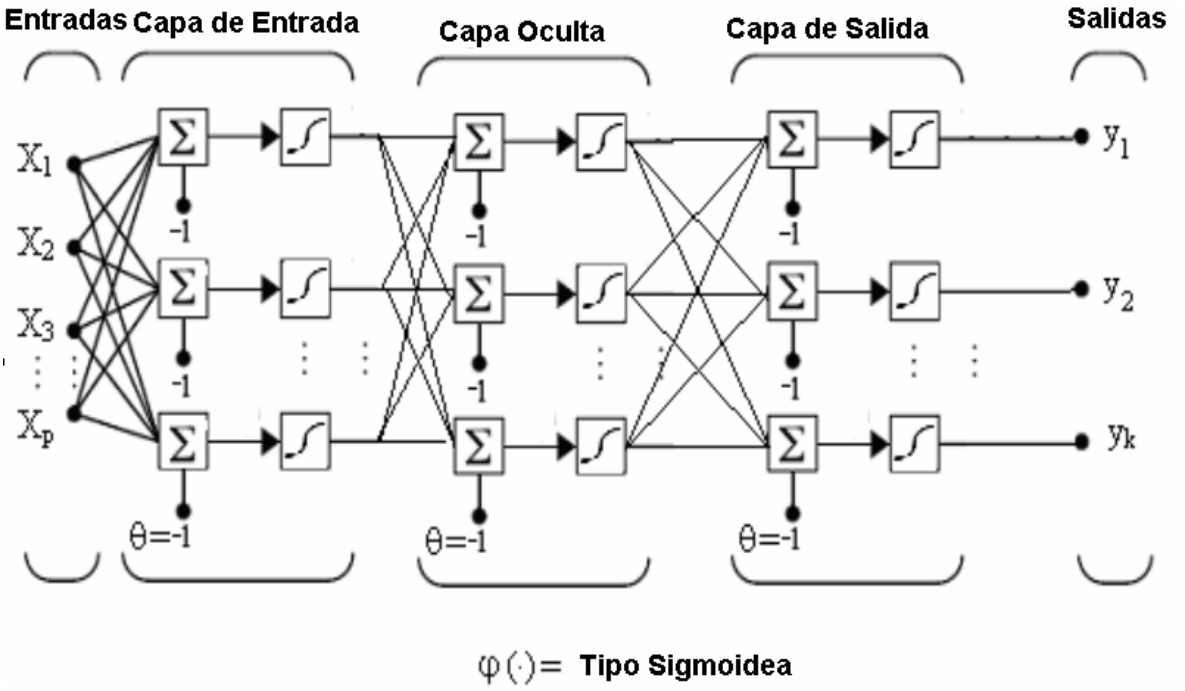


Figura 3.7. Arquitectura del Perceptrón Multicapa

Al observar la arquitectura del Perceptrón Multicapa se puede observar que las múltiples entradas conectadas en la primera capa son mapeadas en las salidas en función de las distintas capas de neuronas intermedias y de los parámetros libres de la red. Se puede ver como una caja negra que realiza una operación sobre las entradas, produciendo un rango de salidas en función de los parámetros libres.

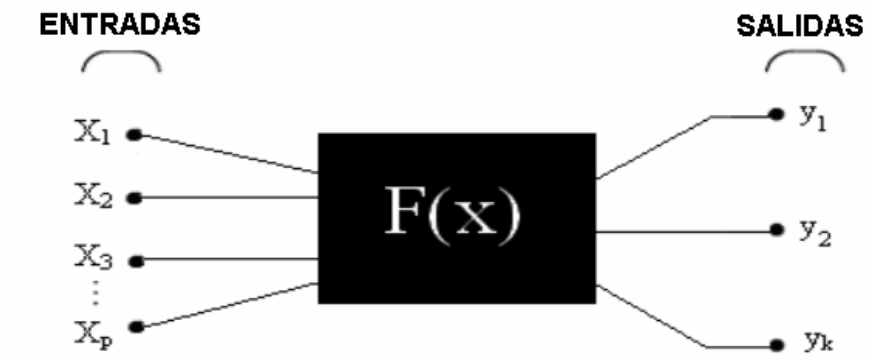


Figura 3.8. Perceptrón Multicapa visto como una caja negra.

Se infiere de la arquitectura que el algoritmo de entrenamiento de una red con tales características deberá ser planeado para que los cambios en los parámetros libres sean tales que el error en las unidades básicas de la estructura sea mínimo de manera que el conjunto de los cambios produzca un error global que tienda al mínimo. Se buscará entonces el límite en el cual la configuración de los parámetros libres produzca un error mínimo.

Tomando en cuenta este razonamiento, la evolución al Perceptrón Multicapa tuvo que basar su éxito en el diseño del algoritmo de entrenamiento que lograra minimizar el error al modificar adecuadamente los pesos y umbrales. La historia marco como primer paso, estudiar la forma de minimizar el error en una capa de neuronas lineales, que se conoce también como filtro lineal. El análisis de este tipo de red neuronal que posee elementos lineales, nos permite deducir un algoritmo más complejo para entrenar a una red como la Perceptrón Multicapa que posee elementos no lineales.

3.2.2 ALGORITMO DE MINIMOS CUADRADOS (LMS).

Imaginemos entonces a la red lineal de la que hablamos con un numero k de neuronas lineales como nos muestra la Figura 3.9

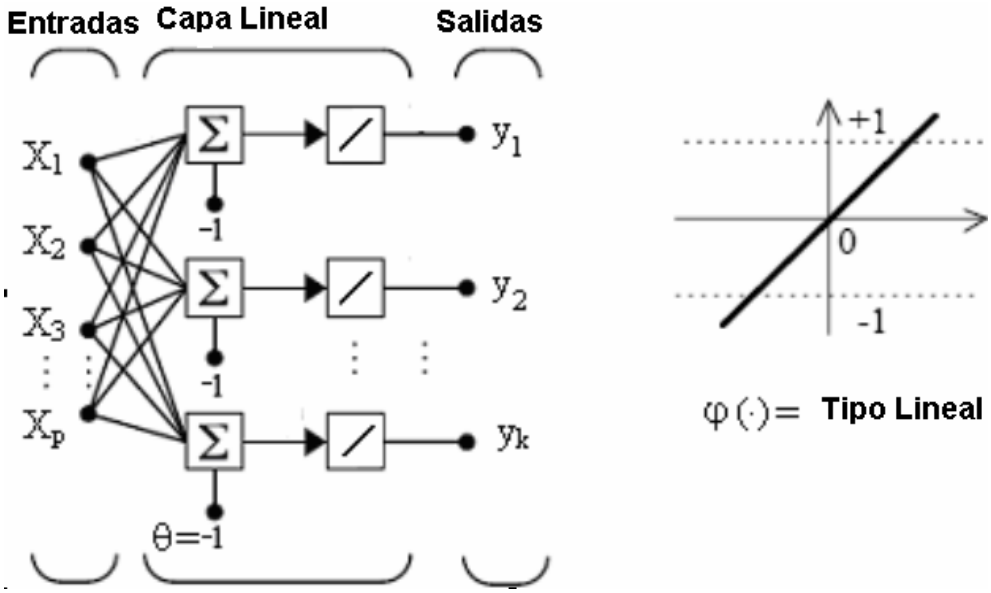


Figura 3.9. Arquitectura de una red lineal.

Se puede notar que a diferencia del Perceptrón las neuronas de esta red tienen una función de activación lineal que le permite a la red tomar valores de salida dentro de un rango continuo.

Se definirá ahora la función de costo J que se utilizará para medir el desempeño de esta red como la media de la suma cuadrada de los errores generados en cada neurona MSE (*mean square error*), como índica la siguiente ecuación.

$$mse = \frac{1}{P}\sum_{k=1}^{P} e(k)^2 = \frac{1}{P}\sum_{k=1}^{P} (d(k) - y(k))^2 \qquad \textbf{(3.5)}$$

Si visualizamos la relación que existe entre los parámetros libres de la red y la MSE que genera cada combinación de parámetros se obtendrá una superficie de error en la cual la configuración de los pesos de la red y los umbrales generará a cada punto un error. La

idea de nuestro algoritmo es hallar el punto mínimo en esa superficie de error, indicando entonces el error mínimo.

Para un filtro lineal, la superficie de error que se genera es un paraboloide o parábola tridimensional [BEA03]. Este hecho nos índica que solo existirá un mínimo en esta superficie ya que las parábolas solo tienen un mínimo como se muestra a continuación en la Figura 3.10.

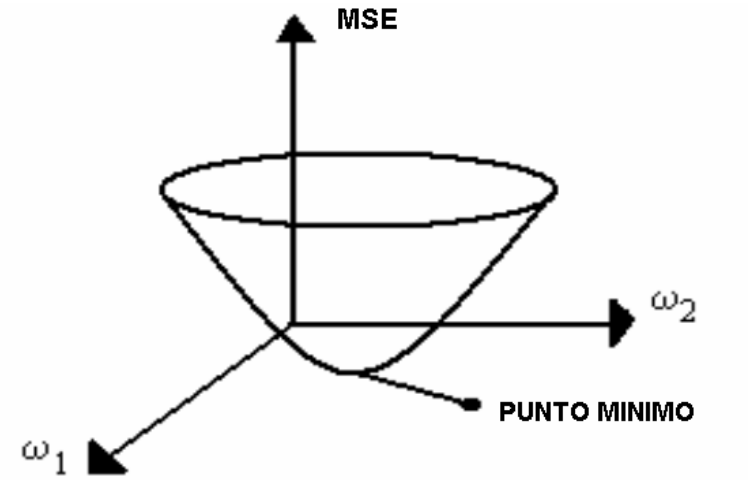


Figura 3.10. Superficie de Error (MSE) de una Red Lineal.

De esta figura se puede inferir que el algoritmo a diseñar deberá, a partir de un punto inicial, rastrear sobre la superficie de error el punto marcado como mínimo hallando los valores óptimos de pesos y umbrales, teniendo en cuenta que la función de costo a minimizar será la MSE. Para determinar dicha combinación óptima se derivará parcialmente la función de costo con respecto a los parámetros libres y se dirá que el punto óptimo será aquel en que la derivada parcial sea cero. Esta derivada parcial es conocida como el gradiente de la función de costo con respecto a los pesos que esta definido como sigue:

$$\nabla_{\omega k} J = \frac{\partial J}{\partial \omega_k}, k = 1,2,... p \quad \textbf{(3.6)}$$

El gradiente es un vector, de dirección perpendicular a las curvas de nivel de la superficie y que indica el sentido de crecimiento de la función. En general a cada punto de la superficie le corresponde un gradiente. La dirección en la que la pendiente de la función de error crece más rápido es la dirección del gradiente, mientras que la dirección en la que decrece más rápido es la dirección negativa del gradiente.

Imaginemos que la superficie de error, descrita como MSE (ω,b), fuese una montaña y que un algoritmo comienza inicializando a un esquiador en una de las cumbres de la montaña. A partir de este punto el algoritmo tratará de que el esquiador descienda en dirección de la trayectoria de máxima pendiente. Si MSE es una función diferenciable en el punto (ω,b) entonces se dirá que si el gradiente en ese punto es cero índica un mínimo en la superficie de error y que la dirección de máximo crecimiento viene dada por $\nabla_{\omega k} J$ y la dirección de máximo decrecimiento viene dada por $-\nabla_{\omega k} J$. Este tipo de método de descenso se conoce como el algoritmo del 'descenso más empinado' o *steepest descent* en ingles, el cual propone ajustar iterativamente los pesos moviéndolos hacia la solución óptima en dirección contraria al gradiente que es la dirección de mayor decrecimiento de manera que el ajuste de los pesos queda de esta manera:

$$\begin{aligned} \Delta\omega_k(n) &= -\eta\nabla_{\omega k} J(n) \\ \omega_k(n+1) &= \omega_k + \Delta\omega_k(n) \\ \omega_k(n+1) &= \omega_k - \eta\nabla_{\omega k} J(n) \end{aligned} \quad \textbf{(3.7)}$$

El método del descenso empinado se dicta como sigue: "El valor actualizado del peso k ésimo de una red lineal es igual al valor anterior de el peso en cuestión más una corrección que es proporcional al negativo del gradiente en la superficie de error con respecto a esos pesos en particular" [HAY94]. En este momento aparece en las ecuaciones la constante η conocida también como coeficiente de velocidad de aprendizaje la cual servirá al algoritmo para atenuar el aprendizaje de la red haciéndolo más fino. A continuación se muestra en la Figura 3.11 el Diagrama a bloques del algoritmo del descenso empinado

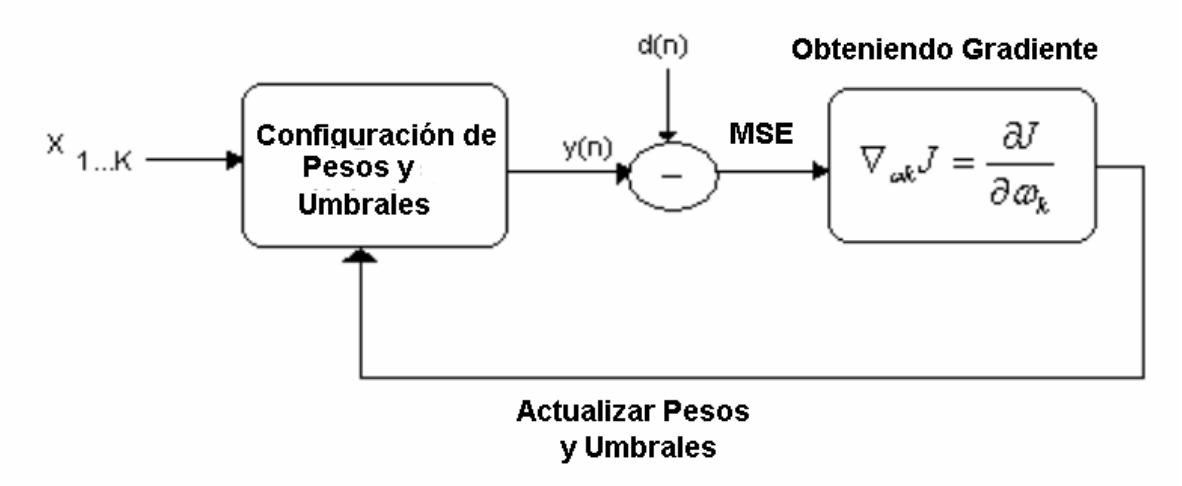


Figura 3.11. Diagrama a bloques del Algoritmo del Descenso Empinado.

El método del descenso más empinado (Steepest Descent) es exacto en el sentido de que no utiliza aproximaciones en sus derivadas. Sin embargo, el cálculo del gradiente, utilizando al MSE como función de costo, requiere conocimiento previo de la función de correlación espacial, en otras palabras este método es bueno cuando se aplica a un ambiente conocido y hace que los pesos sigan una trayectoria determinada a través de la superficie de error, llegando a un peso optimo.

Sin embargo este método no es el más adecuado para hallar un mínimo en una superficie de error. Retomando el problema de la detección de intrusos nos damos cuenta de que si el ambiente fuera estacionario (características estadísticas que no cambian con el tiempo) en teoría la red podría aprender con un método así, sin embargo el flujo de datos en una red de comunicaciones a menudo es impredecible con una estadística estática. No obstante, Widrow y Hoff [HAY94] diseñaron un algoritmo conocido como de Mínimos Cuadrados LMS (*Least Mean Square*), que utiliza una aproximación del método del 'descenso más empinado' para hallar el error mínimo.

Este algoritmo esta basado en el uso del estimado del valor instantáneo de la función de costo. De esta manera el algoritmo esta listo para actuar en un ambiente no estacionario ya que no necesita conocer el entorno sino simplemente hacer una aproximación de el. El LMS se aplica a un ambiente desconocido y el peso estimado sigue una trayectoria estocástica o de movimiento Browniano a través de la superficie de error.' [HAY94]. Así pues la aproximación utilizada para la función de costo es como sigue:

$$\varepsilon(n) = \frac{1}{2}\sum_{k} e^{2}{}_{k}(n) \qquad \textbf{(3.8)}$$

Representando la suma de los errores cuadrados. La red es entonces optimizada minimizando esta función con respecto de los pesos sinápticos de la red. La derivada parcial de $\varepsilon(n)$ con respecto a los pesos queda de la siguiente manera:

$$\frac{\partial e^{2}(k)}{\partial \omega_{j,k}} = \frac{2}{2} e(k) \frac{\partial e(k)}{\partial \omega_{j,k}} \qquad \textbf{(3.9)}$$

Posteriormente la derivada del error e(k) con respecto a los pesos queda como sigue:

$$\frac{\partial e(k)}{\partial \omega_{k,j}} = X_j(k) \tag{3.10}$$

Finalmente el incremento en peso actualizado queda de la siguiente manera:

$$\Delta \hat{\omega}_{kj}(n) = \eta\, e_k(n)\, X_j(n) \tag{3.11}$$

Teniendo en cuenta que $\hat{\omega}$ es el valor estimado del peso. A continuación se muestra el algoritmo del LMS tal como queda finalmente:

1. Inicialización.

$\hat{\omega}_k(1) = 0$
$k = 1,2,.... \; p$

2. Filtrado. Para tiempos n = 1, 2, …, ultima muestra.

$$\begin{aligned} y(n) &= \sum_{j=1}^{p} \hat{\omega}_j(n) X_j(n) \\ e(n) &= d(n) - y(n) \\ \hat{\omega}_k(n+1) &= \hat{\omega}_k(n) + \eta e(n) x_k(n) \\ k &= 1,2,..., \; p \end{aligned} \tag{3.12}$$

El algoritmo de LMS nos índica que la ΔW estará dada por el producto de un coeficiente η de velocidad de aprendizaje o *learning rate*, y el vector de entrada multiplicado por el valor instantáneo de la función de error. Este algoritmo de simple implementación nos da las bases para el diseño de algoritmos de estimación estocástica que nos permitan trabajar en otro tipo de superficies de error como las que generará una red del tipo Perceptrón Multicapa cuyos elementos trabajan en regiones no lineales de manera que un algoritmo determinista sería totalmente inadecuado para hallar el mínimo en la superficie de error, de hecho, el algoritmo que se mostrará a continuación para el entrenamiento de una red Multicapa esta basado en estimaciones como las hechas en LMS.

3.3 ALGORITMO DE PROPAGACION HACIA ATRÁS.

3.3.1 PROPAGACION HACIA ATRÁS CON GRADIENTE DESCENDIENTE.

Retomando la arquitectura del Perceptrón Multicapa de la Figura 3.7 ubicamos un nuevo concepto de sistema neuronal distinto a lo que habíamos estudiado en redes de una sola capa, el Perceptrón Multicapa propone una estructura de capas ocultas que realizan procesamiento complejo sobre las entradas. Además de esto, el uso de neuronas con función de activación no lineal, como lo es la sigmoide, permite a la red aprender las discontinuidades de los entornos.

Toda esta evolución en la arquitectura nos lleva a plantear un nuevo y complejo algoritmo de entrenamiento que permita a una red de este tipo llevar al mínimo una función global de error que es dependiente, no solo de los parámetros libres de la red, sino también de las diversas interconexiones entre los elementos del sistema que desempeñan una labor de conceptualización global y compleja.

La visión que ahora se tiene de los algoritmos de entrenamiento de redes de una sola capa nos permite introducir a este nuevo algoritmo que permite entrenar a redes de muchas capas. El algoritmo de *Back Propagation* o Propagación hacia atrás esta basado en el algoritmo de la regla de corrección de error o también llamado regla delta. En el Back Propagation se calcula el error instantáneo de la suma de los errores cuadrados para derivar la función de costo y depende únicamente de los elementos de la última capa visible. El error instantáneo esta dado por la siguiente ecuación:

$$\varepsilon(n) = \frac{1}{2}\sum_{j \in c} e_j^{\,2}(n) \qquad \textbf{(3.13)}$$

Donde c incluye a todas las neuronas de la capa de salida. Si ahora decimos que N denota el número total de muestras contenidas en un conjunto de entrenamiento, entonces se puede calcular el valor promedio del error cuadrado con la ecuación 3.13:

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N} \varepsilon(n) \qquad \textbf{(3.14)}$$

Así pues la función de costo a minimizar será la anteriormente denotada por la ecuación 3.14. Para minimizar esta función se usará una aproximación similar a la usada para derivar el algoritmo de LMS y nuevamente usando el concepto de gradiente. Para esto calculamos el valor del gradiente instantáneo de la función de costo con respecto a los pesos sinápticos como muestra la siguiente ecuación aplicando la regla de la cadena donde la derivada parcial se descompone en las funciones de las que depende la función inicial:

$$\frac{\partial \varepsilon(n)}{\partial \omega_{kj}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \frac{\partial \upsilon_j(n)}{\partial \omega_{kj}(n)} \qquad \textbf{(3.15)}$$

Donde k,j son los índices que indican la posición de cada neurona dentro del Perceptrón Multicapa. Posteriormente la delta de cada peso se calcula como sigue:

$$\Delta \omega_{kj}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial \omega_{kj}(n)} \qquad \textbf{(3.16)}$$

Y resolviendo la derivada parcial se obtiene que la delta quede de la siguiente manera:

$$\Delta \omega_{kj}(n) = -\eta \delta_j(n) X(n) \qquad \textbf{(3.17)}$$

Donde X(n) representa el vector de entrada para la muestra n y δ representa el gradiente local que depende de la ubicación de la neurona en la red y queda como sigue:

$$\delta_j(n) = e_j(n)\varphi'(\upsilon_j(n)) \quad \textbf{(3.18)}$$

De acuerdo a esta ecuación el gradiente local para una neurona es igual al producto del error que esta neurona genera y la derivada de la función de activación asociada. Para este tipo de red con varias capas nos enfrentaremos a cálculos para dos tipos de neuronas.

En el primer caso la Neurona j es un nodo de salida. En este caso se puede aplicar simplemente la ecuación 3.16 tomando en cuenta que el gradiente local esta determinado por los valores del error y la derivada de la función de activación datos que son conocidos debido a que el error a la salida de cada neurona es conocido y esta determinado por la ecuación 3.12. De aquí inferimos que para las neuronas ocultas el error que se genera es desconocido y por lo tanto se debe emplear otro tipo de análisis que permita al algoritmo determinar el valor adecuado de delta en cada peso para que el error a la salida sea mínimo. Se dice entonces que el error se debe propagar de la capa de salida hacia la capa de entrada, razón por la cual, este algoritmo se conoce como de Propagación hacia atrás o *Back Propagation* (*BP*).

Para el caso de una neurona oculta se aplica nuevamente la ecuación 2.16 sin embargo el gradiente local se calcula de la siguiente forma:

$$\delta_j(n) = \varphi_j'(\upsilon_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad \textbf{(3.19)}$$

En este caso el gradiente es igual al producto de la derivada de la función de activación asociada y la sumatoria, amplificada por los pesos, de los gradientes locales calculados para las neuronas de las siguientes capas conectadas a la neurona j. Se puede decir finalmente que para calcular el error de las neuronas ocultas es necesario determinarlas recursivamente usando el error de todas sus neuronas conectadas [HAY94].

El método estándar para calcular los gradientes locales se conoce como el método del gradiente descendiente ya que los pesos se mueven en dirección del negativo del gradiente sobre la superficie de error. Si nos damos cuenta, el cálculo del gradiente local implica obtener la derivada de la función de activación por lo que es importante que la función de activación sea diferenciable. Esta propiedad hace que el Perceptrón Multicapa sea adecuado para este tipo de entrenamiento debido a que las neuronas de su arquitectura utilizan funciones de activación diferenciables como lo es la función Sigmoide.

Como se puede observar de las ecuación 3.17, la delta de los pesos depende también de un factor conocido como coeficiente de velocidad de aprendizaje η el cuál se mantiene fijo para todas las deltas de manera que se puede controlar la velocidad con que el algoritmo debe converger al punto mínimo.

En este momento es importante resaltar que la superficie de error que se genera por la combinación de pesos, en una red del tipo Multicapa, es completamente distinta a la generada por una red lineal. En este caso la superficie de error tiende a tener más de un mínimo como se puede observar de la Figura 3.12.

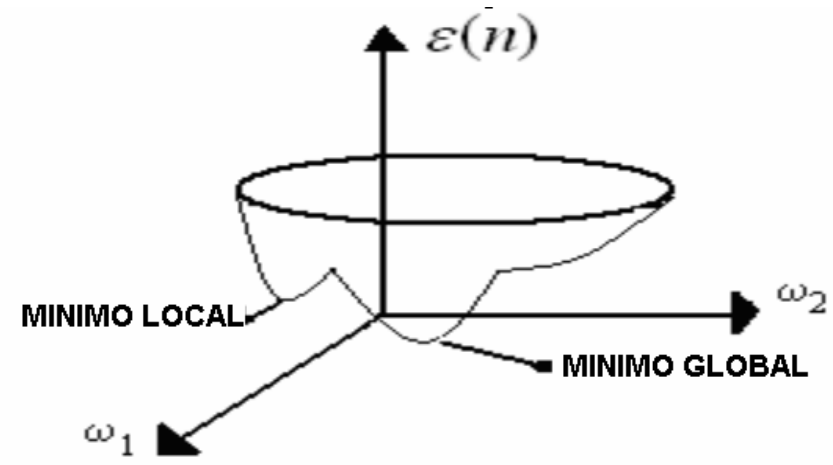


Figura 3.12. Superficie de Error del Perceptrón Multicapa.

Si se tiene esto en consideración es importante que nuestro algoritmo no sea muy acelerado al ir en busca del mínimo ya que podría hacer que el sistema se volviera inestable o se perdiera en un mínimo local.[BIS95] Es decir, que un valor muy grande de η haría que el algoritmo oscile indefinidamente en busca del mínimo global. Teniendo esto en cuenta es importante que el valor de η no sea muy alto para evitar este problema, en la práctica η debería ser dependiente de la conexión con cada neurona ya que algunos pesos se deben ajustar más rápido que otros.

3.3.2 PROPAGACION HACIA ATRÁS CON MOMENTO.

Por otro lado si se quiere evitar inestabilidad por el uso de una η muy grande, se puede agregar un coeficiente de momento al Back Propagation. El momento denotado por α, cuyo rango puede ser cualquier valor entre 0 y 1, permite a la red responder no solo en base al gradiente local sino también a las tendencias recientes en la superficie de error.

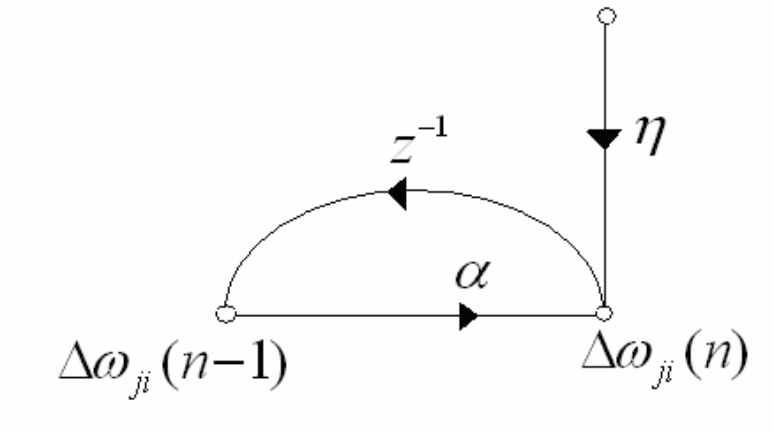


Figura 3.13. Diagrama de flujo del momento.

Como muestra el diagrama de flujo de la Figura 3.13, la ecuación de la regla delta queda como sigue:

$$\Delta\omega_{ji}(n) = \alpha\omega_{ji}(n-1) + \eta\delta_j(n)x_k(n) \qquad \textbf{(3.20)}$$

El momento permite a la red ignorar características pequeñas o de mínimo significado en la superficie de error. Sin el momento una red puede estancarse en un mínimo local superficial, con momento la red puede deslizarse y evitar caer en dicho mínimo [HAY94].

Démonos cuenta que cuando la constante de momento es 0 la delta de cada peso queda determinada únicamente por el gradiente. En cambio cuando el momento es 1, el nuevo cambio de peso se establece igual al anterior cambio ocurrido en el peso y el gradiente simplemente es ignorado. El momento acelera el aprendizaje cuando tiene que hacerlo y lo estabiliza cuando esta cerca de la solución de esta manera previene al algoritmo de caer en un mínimo local en la superficie de error. Finalmente se recomienda que cuando se trabaje con el parámetro de velocidad de aprendizaje η variable, las neuronas con muchas entradas deban tener un parámetro de velocidad de aprendizaje más pequeño que las neuronas con pocas entradas.

3.4 PROPIEDADES DEL PERCEPTRÓN MULTICAPA.

La arquitectura del Perceptrón Multicapa mostrada en la Figura 3.7 presenta un modelo de capas de neuronas cuya función de activación es una función que trabaja en zonas no lineales. Como ya se ha visto con anterioridad es importante que esta función de activación sea diferenciable para poder aplicar el algoritmo de propagación hacia atrás. Hasta este momento solo se conoce la función sigmoide la cual viene representada en la Figura 3.6 y cuya expresión queda de la siguiente forma:

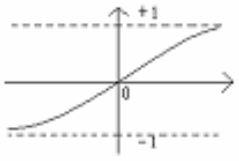
$$\varphi(\upsilon) = \frac{1}{1 + \exp(-\upsilon)} \qquad \textbf{(3.21)}$$

Como muestra la Figura 3.6, las neuronas sigmoideas presentan la mayor pendiente cuando las neuronas trabajan más cerca de su rango medio. Por el contrario, cuando las neuronas trabajan cerca de sus límites la pendiente es mínima y tiende a cero. Esta es la razón por la cual se debe buscar que las neuronas sigmoideas se entrenen con salidas destino entre .1 y .9 y no en 0 y 1 ya que en los límites de la sigmoide, la derivada se aproxima a cero y entonces el cambio de los pesos del algoritmo de propagación hacia atrás será mínimo haciendo mas lento el aprendizaje. 'Sin embargo es totalmente razonable utilizar los límites 0 y 1 como entradas de la red' [MAS93].

La derivada de la función sigmoidea se encuentra fácilmente ya que es:

$$f'(x) = f(x)\,(1-f(x)) \qquad \textbf{(3.22)}$$

La relativamente pequeña derivada de la función sigmoidea hace lento el proceso de aprendizaje en el algoritmo básico de propagación hacia atrás, debido a estas razones se propone el uso de otra función conocida como tangente hiperbólica' [MAS93], cuya gráfica se muestra a continuación:



$$\varphi(\upsilon) = \frac{1-\exp(-\upsilon)}{1+\exp(-\upsilon)}$$

Figura 3.14. Función de Activación Tangente Hiperbólica.

Esta función presenta ciertas ventajas en relación a la sigmoidea debido a que su rango de salida va de -1 a 1, esto permite que la función se considere simétrica ya que cumple con $\varphi(-\nu) = -\varphi(\nu)$ y entonces el Perceptrón Multicapa entrenado con BackPropagation aprende más rápido en términos del número de iteraciones requerido.

No obstante siempre se recomienda que se pueda variar la función de activación para esperar resultados distintos. De manera que cada problema encuentre la arquitectura neuronal que más le convenga.

En el caso de que se intente obtener modelos de funciones por medio de la red neuronal, es decir, una aproximación de funciones lineales o no lineales es importante tener en la capa de salida neuronas lineales y usar funciones de activación de tangente hiperbólica en las demás capas. Esto es debido a que las neuronas de salida lineal nos permiten acceder a rangos de salida distintos a cero y uno y de esta manera la aproximación a la salida será más cercana al destino deseado. Las neuronas no lineales en las capas ocultas permitirán a la red aprender las no linealidades que posiblemente se encuentren en la función a aproximar. Para el problema de la detección de intrusos no se esta haciendo un modelado de función sino mas bien una clasificación de patrones por lo que probablemente sea conveniente que todas las neuronas sean no lineales.

Por otro lado siempre se puede topar con el problema de la saturación incorrecta de las neuronas, esto sucede cuando el algoritmo de entrenamiento se estanca en un mínimo local o superficial de manera que las neuronas se saturan antes de tiempo provocando que trabajen cerca de los límites de la función que es justo donde la pendiente se aproxima a cero por lo que los cambios en los pesos ocurrirán mínimos.

‘Para evitar este problema se recomienda que los pesos se inicialicen uniformemente distribuidos dentro de un rango pequeño para evitar que la función de activación se dispare a zonas de saturación. Además de esto, será importante mantener bajo el número de neuronas ocultas debido a que el uso de neuronas extras puede acarrear consecuencias graves como que la red aprenda idiosincrasias en los patrones evitando un

conocimiento acertado y se debe procurar trabajar a las neuronas en la zona media donde se considera que la región es lineal [HAY94]. Se recuerda que un buen ajuste es logrado con el modelo de red más simple adecuado al problema.

A continuación se hablará de dos formas en las que se puede entrenar una red neuronal en cuanto a la presentación de los datos a la red y los ajustes que se llevan a cabo, en el primer caso se tiene al entrenamiento de *patrón por patrón*, en el cuál a la red se le presenta un patrón e inmediatamente después se realiza la propagación hacia atrás ajustando los pesos. Este modo de entrenamiento es útil debido a que el ajuste de los pesos sucede aleatorio debido a que con cada patrón surge un nuevo ajuste de pesos, de esta manera se evita al algoritmo de caer en un mínimo local y ocupa menos memoria, ya que no almacena los cambios. En cambio el otro caso de entrenamiento corresponde al modo de conjunto de datos o *Batch,* en el cual los datos se presentan a la red y el calculo de la modificación en los pesos se almacenan hasta que todas las muestras han sido presentadas a la red, en este momento se modifican los pesos con la suma de las modificaciones almacenadas. Este algoritmo es bueno debido a que estima de manera más precisa el vector gradiente y es más eficiente computacionalmente. En general se maneja que el método de Batch es el más adecuado para los problemas de clasificación y detección de patrones como lo es el problema en cuestión.

Si decimos que a la presentación de todo el conjunto de muestras a la red neuronal se le llama epoch, se recomienda para cualquiera de los casos mencionados anteriormente que el orden de las muestras sea aleatorizado de epoch en epoch para aumentar la velocidad de convergencia [BIS95].

Y en este momento destacamos la verdadera importancia que tiene el hecho de que se le permita al algoritmo converger rápidamente hacia un mínimo. La cuestión de la velocidad de convergencia es importante debido a que nos representará una menor perdida de tiempo al entrenar la red, y por otro lado asegurará de que la red trabaje correctamente con el conjunto de datos de entrenamiento mostrado. Sin embargo se debe aclarar que el hecho de que el algoritmo converja a un valor pequeño no implica una buena generalización [HAY94].

Las redes del tipo Perceptrón Multicapa que se entrenan apropiadamente tienden a ofrecer razonables respuestas cuando son puestas a prueba con entradas que nunca han visto. Esta propiedad de generalización hace posible entrenar a la red con un paquete representativo de pares de entradas y destinos esperados y obtener buenos resultados sin necesidad de entrenar a la red con todos los pares posibles de entradas y salidas.

Se puede ver a la red neuronal como un elemento para hacer mapeo no lineal de entradas y salidas en un problema de ajuste de curva, si imaginamos que contamos con un número infinito de muestras, el error al evaluar la red, es mínimo, sin embargo la idea es que con pocas muestras se pueda lograr la generalización. No obstante, la generalización no siempre se logra aunque la reducción del error siempre se puede lograr. Debido a que la idea es que la red tenga una buena generalización, se determinarán los factores que influyen en esta tarea y que son el tamaño y eficiencia del conjunto de datos de entrenamiento, la arquitectura de la red y la complejidad física del problema. El último punto no se puede controlar, sin embargo, los dos primeros sí.

Una vez que la arquitectura de la red ha sido fijada de acuerdo a la complejidad física del problema se puede utilizar la ecuación 3.23, que propone Haykin [HAY94], para determinar el tamaño del conjunto de datos y que nos dice que esta determinado por:

$$N >= \frac{W}{\varepsilon} \quad \textbf{(3.23)}$$

Donde W es el número de pesos de la red y E es la fracción de error permitido que generalmente se puede establecer en .01, de manera que se puede asegurar que el número de muestras permitirá a nuestra red generalizar correctamente.

3.5 COMENTARIOS FINALES DEL TERCER CAPITULO.

Una de las ventajas en el uso de redes neuronales para la detección de anomalías es que el diseño de la red neuronal esta basado directamente en datos reales. Dejando que los datos hablen por sí mismos, cambiando la visión clásica de diseñar en base a modelos y más bien hacerlo en base a señales de la realidad. Sin embargo el diseño de la red neuronal depende en su totalidad del problema que se este manejando debido a que las arquitecturas pueden variar en cuanto a forma y modo de operación y no existe, al momento, un modelo general que se pueda aplicar a todos los problemas.

Al hablar de detección de anomalías se infiere que es un problema de clasificación de patrones debido a que la red a diseñar debe clasificar un comando del protocolo HTML en el grupo de datos normal o anómalo. Así pues el diseño de la red Perceptrón Multicapa, enfocada a la detección de intrusos, debe estar basado en las normas de diseño de un sistema de clasificación de patrones.

Como se ha visto en este capítulo, la arquitectura de la red neuronal puede variar de acuerdo al problema. Sin embargo, nos dimos cuenta de que el uso de neuronas del tipo

tangente hiperbólica es más eficiente que el uso de neuronas sigmoidales, no obstante, el número de neuronas en la capa oculta sigue siendo una cuestión determinada únicamente por el problema que se esta tratando. Se debe tener en cuenta que las neuronas ocultas trabajan como extractores de patrones globales ya que combinan las salidas de las neuronas de la primera capa operándolas en una región particular del espacio de entrada. Sin embargo el uso de neuronas de sobra puede acarrear problemas de mala generalización y por el contrario la falta de neuronas ocultas puede acarrear la perdida de caracterización.

Incluso el número de capas ocultas sigue siendo un factor a determinar únicamente por el problema en cuestión. Generalmente un problema de detección de patrones como el que estamos tratando puede trabajar hasta con dos capas ocultas debido a que existen muchas características de los datos que una sola capa pudiera no notar, y por lo tanto una segunda capa puede realizar un mejor trabajo.

Otro factor, antes mencionado, a tomar en cuenta es la inicialización de los pesos cuya distribución debe ser uniforme para que las neuronas no tiendan a saturarse antes de tiempo. Se recuerda que las neuronas siempre trabajaran mejor cuando se trabajan en la región media ya que el mayor cambio en la pendiente se presenta en esta zona.

Por último el tamaño del conjunto de datos deberá prepararse de tal forma que la cantidad de muestras sea por lo menos 10 veces mayor al número de parámetros libres de la red manteniendo así la proporción del 10% de error esperado como lo índica la ecuación 3.23.

A lo largo de este proyecto de tesis se tratará de hallar cual es la mejor arquitectura para el problema en cuestión. Tomando en cuenta todas las recomendaciones de diseño que serán presentadas en el siguiente capítulo enfocándonos a la clasificación

binaria. Cabe resaltar finalmente que este proyecto buscará acercar el mejor modelo de red neuronal que sea eficiente para detectar patrones en el flujo de datos de redes de computadoras y para esto en la segunda parte destinada a los resultados prácticos se simulará a la red con el mayor número de variaciones posibles en la arquitectura para observar en que tipo de modelo se observa la mejor eficiencia y así arrojar resultados que sean significativos como principios de diseño al tratar de implementar un sistema de detección de intrusos para trabajos posteriores.