CAPITULO 5 DISEÑO DE UN CLASIFICADOR DE PATRONES.

En este capítulo se presenta un problema simple de clasificación de patrones, en el cual se pondrán a prueba los conocimientos obtenidos en la primera parte de adaptación teórica,

5.1 SISTEMA DE PRUEBA: CLASIFICACION DE LETRAS.

A continuación se llevará a cabo el diseño e implementación de un sistema capaz de discernir entre letras escritas a mano y elegir a que tipo de letra corresponde cada carácter escrito. A menudo es útil un sistema de esta naturaleza en particular si este sistema es capaz de identificar símbolos de manera eficaz. Por ejemplo, una maquina que pueda leer cheques de banco puede procesar muchos mas cheques de los que un humano podría en el mismo tiempo.

En un principio se puede decir que una letra escrita a mano puede ser considerada un modelo ideal de tipo de letra pero con ruido. Es decir que se puede representar las variaciones en la escritura de las letras como ruido en las mismas letras. Ya que al final todas las formas se acercan a un modelo ideal solo que con pequeñas variaciones del modelo original. Partiendo de esta premisa se continuará con el proceso de entrenamiento de nuestras redes neuronales tomando en cuenta que existen por lo general cuatro pasos en este proceso que son:

1. Ensamblar el paquete de datos de entrenamiento.
2. Crear y definir a la red.
3. Entrenar a la red.
4. Simular la respuesta de la red con nuevas entradas [HAY94].

El paquete de datos con el que contamos es un conjunto de 26 vectores binarios que representan cada una de las 26 letras del alfabeto en su modelo ideal. Es decir, el modelo ideal con el cual se entrenará a nuestra red. En la Figura 5.1 observamos el modelo de la letra A como un vector binario de 35 posiciones representando las 35 celdas que representan matricialmente a la letra A tomando en cuenta que cada letra se represente en un rectángulo de 5 bits de ancho por 7 bits de alto. Todas las letras serán representadas de esta forma por lo que el conjunto total de datos estará definido por una matriz binaria de 26 X 35 representando las 26 letras del alfabeto en 35 bits.

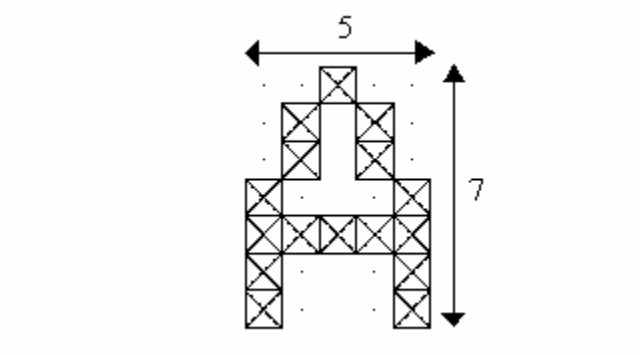

Figura 5.1. Modelo de la letra A en 35 bits.

Esta letra se encuentra representada por el siguiente vector de 35 bits. [00100 01010 01010 10001 11111 10001 10001], que como se verá posteriormente corresponde a un vector de entrada para el entrenamiento de nuestra red neuronal.

A continuación se verá como se definirán y entrenarán las distintas redes neuronales para obtener la mejor solución para este problema de clasificación de patrones.

5.2 DEFINICION Y ENTRENAMIENTO DE LA RED NEURONAL.

En esta etapa práctica del proyecto, se utilizarán las herramientas que nos proporciona el toolbox de MATLAB ® destinado a redes neuronales. Este paquete de herramientas nos permite definir, entrenar y simular cualquier tipo de red neuronal y es por esto que a lo largo del proyecto será utilizado extensivamente para obtener el modelo de red neuronal más adecuado para solucionar el problema en cuestión.

Para este caso práctico específico, MATLAB® nos provee de un conjunto de datos de entrenamiento que contiene la matriz de 26 caracteres cada uno representado por su vector de 35 valores binarios [BEA03]. Además de esto, nos provee de los destinos necesarios para identificar a que tipo de letra corresponde cada vector de 35 bits. Es decir, que se tendrá una matriz de destinos cuya única finalidad será representar las 26 salidas posibles que nuestra red neuronal deberá tener. La Figura 5.2 nos muestra la matriz de destinos que MATLAB nos permite usar:

Figura 5.2. Matriz de destinos, 26 salidas por 26 letras.

En este momento contamos con las 26 entradas posibles y los correspondientes 26 destinos de manera que estamos listos para entrenar la red neuronal. El primer paso es llamar al conjunto de datos por medio de la siguiente instrucción:

[alphabet,targets] = prprob;

Con la que se guardará la matriz de entrada en la variable alphabet y la matriz de destinos en la variable target. Posteriormente se deberá definir la red neuronal, la cual será objeto de entrenamiento, por medio de la siguiente instrucción en MATLAB ®:

S1=13 ← 13 Neuronas Ocultas; el número de neuronas en la capa oculta ha sido determinado por prueba y error, mostrando la mayor eficiencia.

S2 = 26 ← 26 Neuronas de Salida.

net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');

Donde definimos que se creará una nueva red del tipo Perceptrón Multicapa por medio de la instrucción newff y los argumentos definirán el tipo de red que se ocupará. El primer argumento minmax(alphabet) hace referencia a los valores mínimos y máximos del conjunto de datos de entrada, las variables S1 y S2 hacen referencia al número de neuronas en la capa oculta y de salida respectivamente, las variables ‘logsig’ ‘logsig’ indican el tipo de neuronas que se usarán en cada capa que en este caso nos indican funciones de activación del tipo sigmoidal. Y por último el argumento traingdx nos indica el tipo de entrenamiento que se efectuará en dicha red el cual para este caso es el entrenamiento delta-bar-delta con el cual queda definida nuestra red, y almacenada en la variable net. A continuación se lleva a cabo el entrenamiento de dicha red neuronal por medio de la siguiente instrucción:

[net]= train(net,P,T);

La cual nos devuelve una red entrenada con el vector de entradas P ajustado a los destinos de la variable T por medio del algoritmo de entrenamiento seleccionado anteriormente en la definición de la red.

El proceso de entrenamiento arroja la siguiente gráfica en la cual se puede apreciar como desciende el error ajustando los pesos para obtener la combinación óptima de pesos y error mínimo.

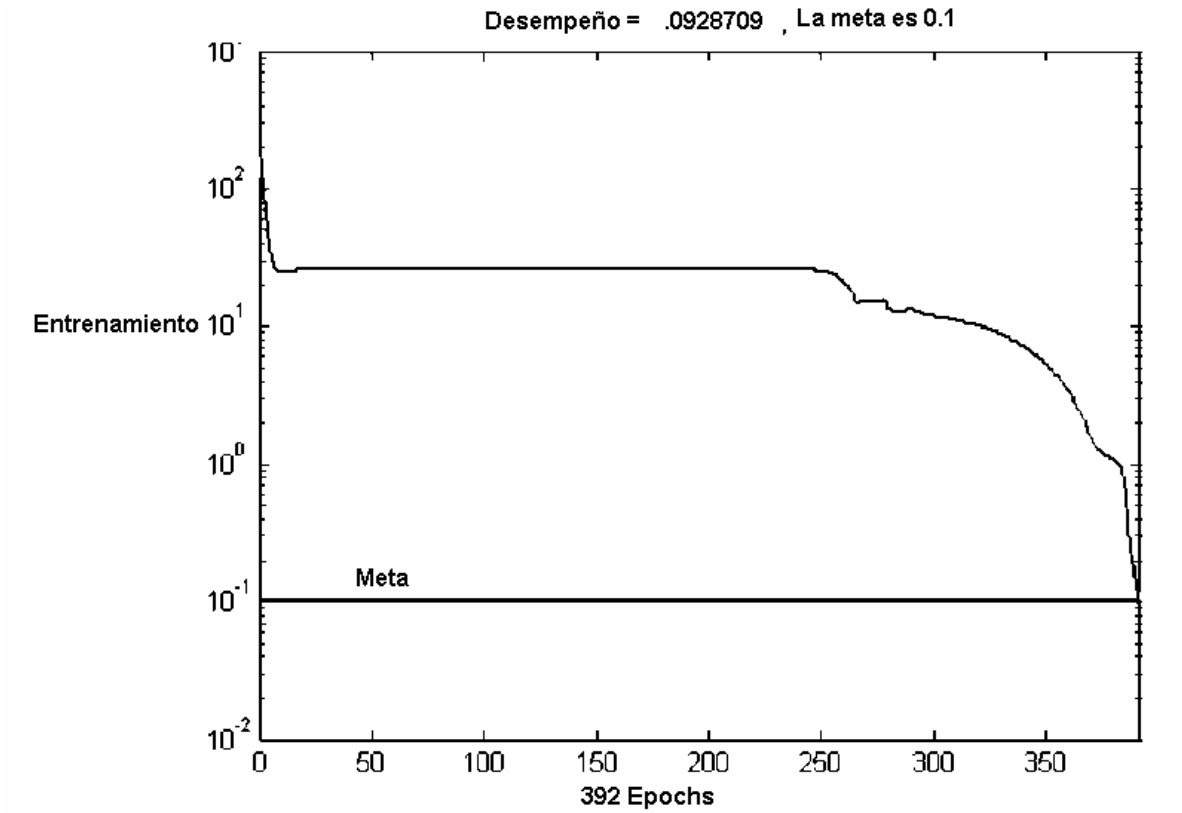

Figura 5.3. Entrenamiento de la Red Neuronal.

Al momento en que el error alcanza el valor mínimo esperado, se detiene el entrenamiento y los pesos de la red quedan ajustados para obtener este error mínimo cuando se le presenten los datos de entrenamiento.

Sin embargo, como se ha explicado anteriormente, la importancia de las redes neuronales no se encuentra únicamente en aproximar los datos con los que ha sido entrenada, sino generalizar este error mínimo para datos de la misma naturaleza que nunca halla visto la red, o a los que nunca haya sido expuesta, de manera que la red pueda efectuar la clasificación adecuada en base a los conocimientos que previamente tiene del modelo ideal de datos con el que fue entrenada.

El interés de esta etapa práctica es que una vez que la red ha sido entrenada con los datos del modelo ideal de las letras, la red pueda a su vez identificar las letras que se acerquen al modelo ideal pero que no sean exactamente este modelo. Para esto la red neuronal que acabamos de entrenar ha sido probada con letras que contienen ruido desde 5% hasta el 50 %. Y así se ha evaluado su funcionamiento.

Sin embargo no ha sido el único modelo de red neuronal que se ha puesto a prueba. A continuación se ha entrenado esta misma red neuronal con letras que contengan ruido de hasta el 20% de manera que mejoramos la calidad del conjunto de datos de entrenamiento, ya que también la entrenamos para asimilar ruido aleatorio como el que podría enfrentar una vez que se ponga a prueba. El ruido aleatorio lo se ha generado con la siguiente instrucción.

*P = [alphabet + randn(R,Q)*0.2];*

Como se puede observar nosotros controlamos la cantidad de ruido que se quiere agregar al conjunto de datos de entrenamiento, y con estos nuevos datos fue re-entrenada la red por lo que al ponerla a prueba se han observado mejores resultados como nos muestra más adelante la gráfica de desempeño de la Figura 5.4.

Además de esta red Multicapa, se ha usado una red Elman para solucionar este mismo problema. Una vez más, el número de neuronas en la capa oculta ha sido determinado por prueba y error, mostrando la mayor eficiencia. La red Elman se define con los mismos argumentos que la red Multicapa únicamente cambiando la instrucción de inicialización por newelm:

```
S1 = 13;

S2 = 26;

netel = newelm(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

El mismo proceso de entrenamiento se llevo a cabo para esta red neuronal, y nuevamente se re-entreno con datos que contenían ruido aleatorio. De esta forma contamos actualmente con 4 redes neuronales entrenadas. Las dos primeras redes son Perceptrón Multicapa una entrenada únicamente con el modelo ideal y la otra con ruido del 20 % y las otras dos son redes del tipo Elman una de las cuales esta entrenada únicamente con el modelo ideal de las letras y la segunda entrenada con ruido del 20%. La gráfica que nos muestra la proporción de error de cada una de estas redes contra la cantidad de ruido de los datos se muestra a continuación en la Figura 5.4.

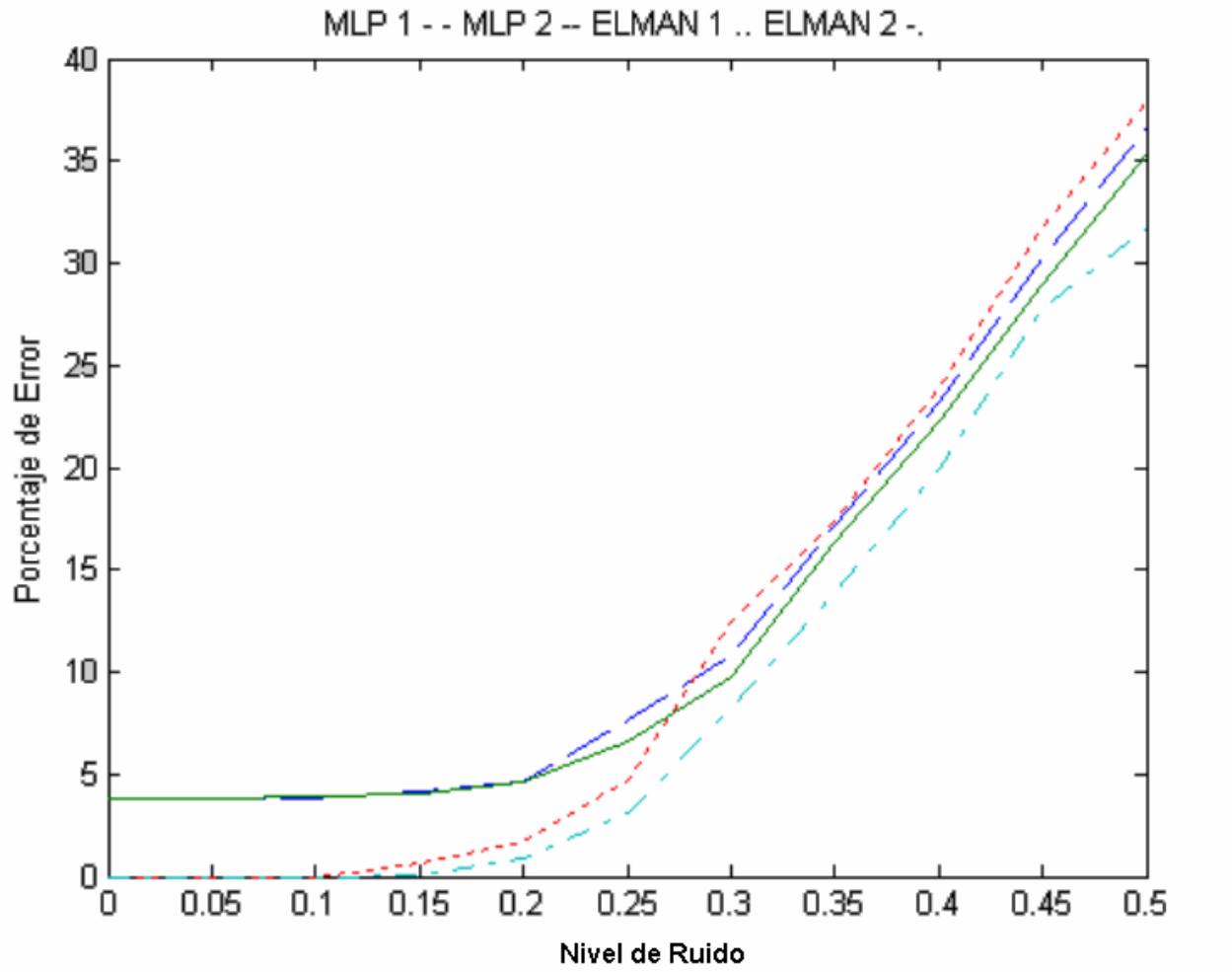

Figura 5.4. Desempeño de las Redes.

5.3 RESULTADOS.

Como se puede observar de la Figura 5.4, en este problema particular de clasificación de patrones, las redes neuronales que fueron entrenadas con ruido del 20% aprendieron a generalizar mejor que aquellas que no fueron entrenadas con ruido. Sin embargo, como se puede apreciar, las 4 redes fueron capaces de generalizar eficientemente para todos los niveles de ruido. Sin embargo la que tuvo un mejor desempeño para altos niveles de ruido

fue la red neuronal Elman, entrenada con ruido ya que obtuvo el menor porcentaje de errores. Muy cerca de ella, se encuentra la red Perceptrón Multicapa que alcanzo un muy buen desempeño en casi todos los niveles de ruido.

De los resultados obtenidos se puede concluir, que la red neuronal Elman presenta una mejora significativa en la solución del problema de clasificación de patrones, debido a su capacidad de retroalimentación que le permite extraer patrones significativos de los datos mejorando su desempeño. No obstante, las redes Perceptrón Multicapa demostraron ser altamente efectivas para los problemas de clasificación evitando que las podamos descartar como herramientas poderosas en la solución de un problema de esta naturaleza.

Como se podrá ver en la siguiente parte, las dos redes fueron probadas con distintos algoritmos de entrenamiento para adecuar una solución al problema de la detección de intrusos, mostrando resultados similares a los obtenidos en esta etapa.

De hecho, el diseño de este sistema clasificador de letras, sienta las bases de diseño para el problema de la detección de intrusos, ya que aunque los dos problemas son totalmente distintos, el enfoque de solución es similar ya que los datos de entrenamiento se pueden tratar de manera muy similar a como han sido tratados en esta sección, salvo ciertas etapas de preprocesado que serán vistas más adelante.

El script completo del código utilizado en este capítulo para el diseño del sistema clasificador de caracteres escritos, se encuentra disponible en el disco compacto anexado como código fuente de este proyecto. A continuación se presenta el diseño del sistema de detección de intrusos que posea las características de generalización que necesitamos de un sistema como este, para la detección de nuevos ataques en el protocolo de comunicación *HTTP*.