CAPÍTULO 4

IMPLEMENTACIÓN DE LA TRANSFORMADA DE DISTANCIA CON UN PERCEPTRÓN MULTICAPA EN MATLAB

4.1 Matlab

Matlab ha demostrado ser un programa de desarrollo muy completo en distintas áreas, más que nada en las tecnológicas. Contiene un ambiente de programación similar a C++ y Java, incluso permite la programación y depuración en estos lenguajes, además de incluir una gran variedad de *toolboxes* que son programas ya desarrollados que facilitan y ahorran la programación al incluir un sin fin de aplicaciones, entre ellas las redes neuronales y el procesamiento de imágenes [8, 9, 13].

Matlab ya trae consigo funciones de redes neuronales en su *Neural Network Toolbox* e incluso cuenta con una interfaz gráfica que facilita ampliamente el diseño de redes neuronales. Al usar esta *toolbox* ya no hay que preocuparse por hacer el programa de una red neuronal, sólo hay que contar con los *patterns*, las características de la red (número de entradas/salidas, rango de valores entradas/salidas, capas ocultas y número de neuronas, funciones de activación), el algoritmo de entrenamiento y los parámetros de entrenamiento (MSE, α, η, μ) [8]. Para llamar a la interfaz gráfica de redes neuronales se ejecuta el comando *nntool* en la ventana principal de Matlab [8]. La Figura 4.1 muestra la ventana principal de la interfaz gráfica de redes neuronales.

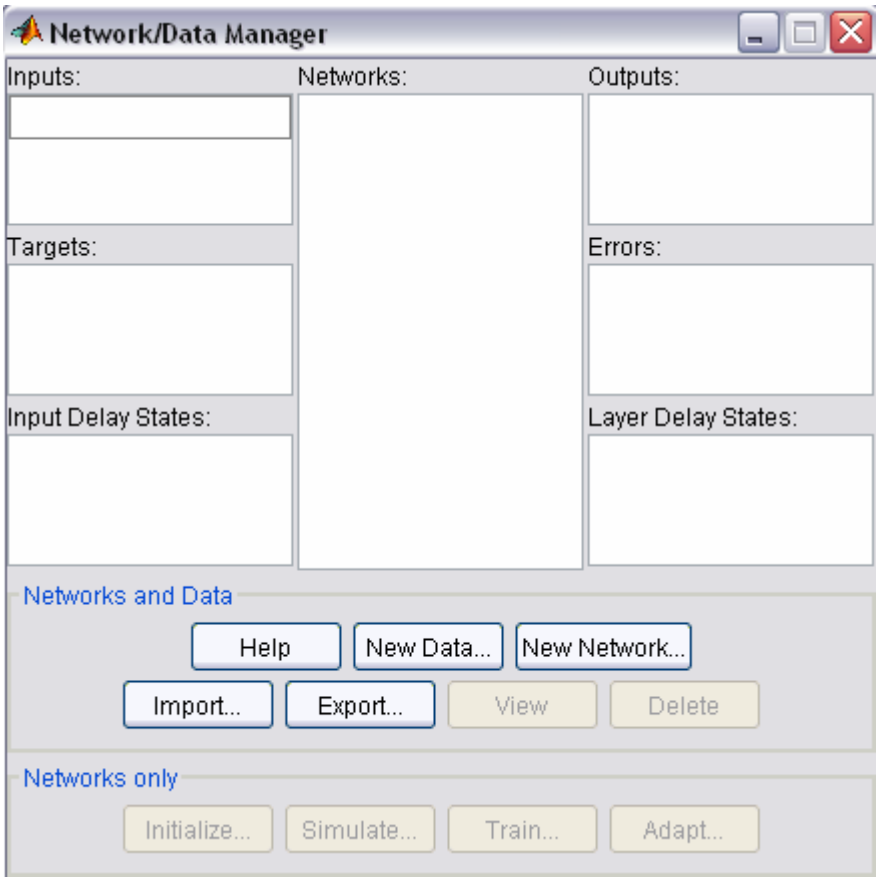


Figura 4.1. Ventana principal de la interfaz gráfica de redes neuronales.

La interfaz gráfica de redes neuronales permite diseñar redes, visualizarlas, entrenarlas, simularlas, y más opciones muy fácilmente, sólo hay que definir los *patterns* (*inputs/targets*), la red (número de entradas/salidas, capas ocultas y número de neuronas por capa), y cambiar a nuestra conveniencia algunos parámetros que trae por *default* para cada tipo de red. Una explicación más amplia sobre el uso de esta interfaz viene en el menú *help* de la ventana principal de Matlab.

Una vez que están definidos todos los aspectos necesarios del *Neural Network Toolbox* ya sea en la interfaz gráfica o con comandos en la ventana principal de Matlab, al

empezar a entrenar una red se nos muestra automáticamente la gráfica del progreso del MSE avanzando en cada *epoch* con la opción de detener el entrenamiento que es excelente más que nada cuando se observa una divergencia o convergencia demasiado lenta.

Matlab tiene varios tipos de redes neuronales en su *toolbox* que vienen perfectamente documentadas en su menú *help*. En este trabajo no se tiene por objetivo ningún tutorial sobre Matlab, sólo la explicación de los comandos utilizados y el programa desarrollado en general. Lo que si es tratado aquí, son los comandos que se deben utilizar para crear una red neuronal en un programa, que es lo que se explica en seguida.

Para crear una nueva red neuronal multicapa se llama a la función *newff* () de la siguiente forma:

net=newff(PR,[S1 S2...SN],{TF1 TF2...TFN},BTF,BLF,PF)

donde net es el nombre deseado para la red, PR es una matriz de R x 2 con los valores mínimo y máximo que cada entrada R puede tomar, SN es el número de neuronas de la capa N (N es el número de capas incluyendo la de salida), TFN es el nombre de la función de activación de las neuronas de la capa N, BTF es el nombre del algoritmo de entrenamiento deseado, BLF es la ecuación de actualización de pesos deseada, y PF es la función de minimización del error que se quiere usar [8].

Para el procesamiento de imágenes, Matlab cuenta con la *Image Processing Toolbox* que tiene funciones para leer imágenes, hacer conversiones de color, y detectar

contornos que son las que más interesan en el desarrollo de este trabajo. Otros procesamientos de imágenes y todos bien documentados se pueden encontrar en el menú *help* del escritorio principal.

4.2 Algoritmo

Como es sabido, siempre se debe tener claro el problema a tratar y las partes en las que se puede dividir para llegar más rápido a su solución. A continuación se enlistan los pasos que definen el propósito principal de este trabajo:

- Leer una imagen de 64x64 píxeles.
- Encontrar el contorno de la imagen.
- Sacar la transformada de distancia de la imagen.
- Implementar la transformada de distancia con un perceptrón multicapa.
- Leer otra imagen de 64x64 y resolver si se trata de una imagen similar.

4.2.1 Leer Imagen y Calcular su Transformada de Distancia

En Matlab es fácil leer una imagen y pasarla al escritorio principal para que pueda ser procesada. El comando es el siguiente:

image=imread('C:\destino donde se encuentra la imagen\nombre de la imagen.extensión');

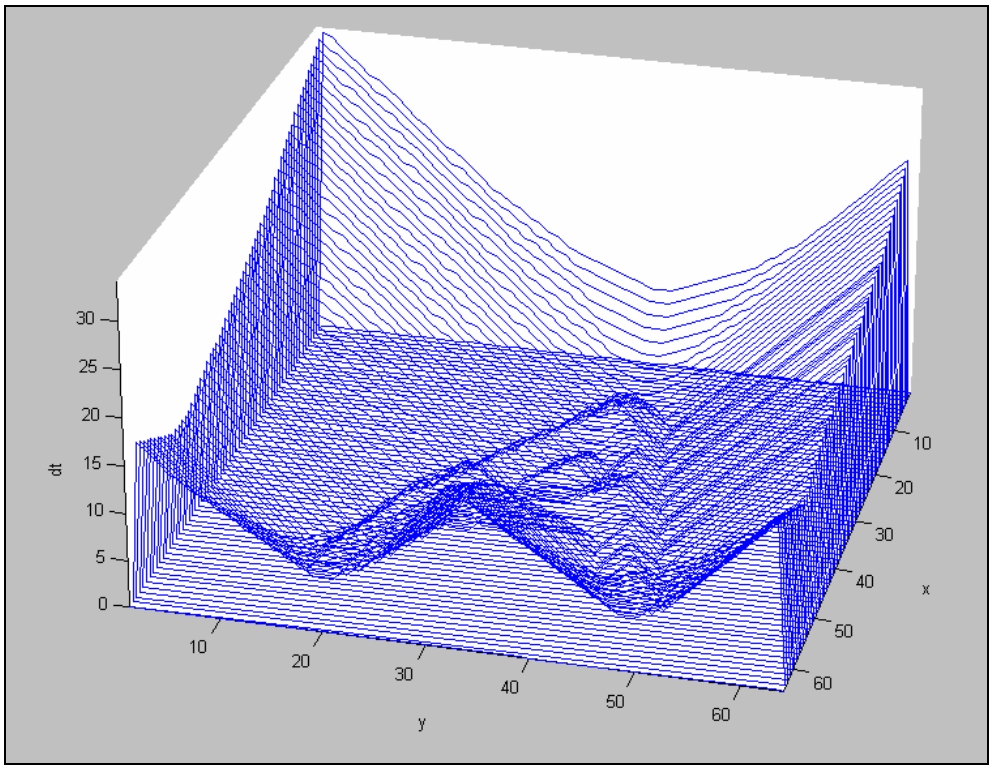
donde *image* es el nombre deseado para guardar la imagen en Matlab, *destino donde se encuentra la imagen* es la carpeta donde se tiene guardada la imagen, *nombre de la imagen.extension* es el nombre con que se tiene guardada la imagen y su formato. El destino donde se encuentra la imagen debe estar dentro de las direcciones que reconoce Matlab. Para agregar una dirección existe una interfaz gráfica fácil de usar siguiendo el menú principal *File-Set Path*. Matlab trae por *default* la carpeta *work* para guardar todo lo que se usa y se crea con este *software*. Cabe mencionar que Matlab soporta diversos formatos de imágenes; por comodidad aquí se ha usado el formato .jpg con imágenes *grayscale*.

En el Apéndice A se describe el programa implementado para calcular la transformada de distancia de una imagen. En este programa se aplica primero la detección del contorno de la imagen, y a este se le hacen los procesamientos necesarios para proseguir al calculo de su transformada de distancia.

La Figura 4.2 (a) muestra una imagen simple del tipo que se usan en este trabajo, en la Figura 4.2 (b) se muestra el contorno obtenido mediante el método de *canny*, y en la Figura 4.2 (c) se muestra el aspecto que tiene su transformada de distancia calculada. En la Figura 4.2 (c) se puede observar muy claramente que la transformada de distancia de una imagen se puede tratar como una función continua a pesar de que en su cálculo sólo se usan valores discretos que corresponde a las coordenadas que definen cada píxel.

(a)

(b)



(c)

Figura 4.2. (a) Imagen original donde el objeto es una letra A. (b) Contorno detectado mediante el método *canny*. (c) Visualización de la transformada de distancia como una función continua.

4.2.2 Entrenar Perceptrón Multicapa

Ya que se tiene la transformada de distancia de una imagen, se sigue con el entrenamiento de un perceptrón multicapa que sea capaz de aprenderla y generalizarla con un error óptimo.

Como se ha mencionado anteriormente, no existe una solución única para la arquitectura de red neuronal que se debe usar en determinado problema, el diseño de redes neuronales es más un arte que una ciencia [3], especialmente al escoger el número de capas ocultas, el número de neuronas en cada capa oculta, y las funciones de activación a usarse. Estos factores dependen enteramente del diseñador, aunque hay reglas que surgen más que nada de los resultados obtenidos en otros problemas.

Como se expuso en el Capítulo 3, está la base teórica del *Universal Approximation Theorem* como sustento de que casi cualquier función continua puede aproximarse mediante un MLP con una capa oculta, pero este teorema no dice nada sobre el tiempo requerido de entrenamiento ni si la implementación es fácil y óptima. Con base en este teorema han surgido métodos empíricos para determinar el número de neuronas ocultas que debe tener un MLP según el problema a resolver, pero no siempre se obtienen buenos resultados [3, 8].

Para la aproximación de funciones, que es el caso de este trabajo, es común usar dos capas ocultas, la primera con menor número de neuronas (aproximadamente la mitad) que la segunda [5, 8], ambas con función de activación sigmoidal ya que se logran relaciones no lineales entre entradas y salidas, y la capa de salida con función de activación lineal para poder obtener cualquier valor real como salida. La primera capa oculta es la que se encarga de extraer las características más importantes de los nodos de entrada, y la segunda capa se encarga de extraer las características de los nodos de entrada en conjunto [3]. Entre más neuronas se tengan en las capas ocultas, mejor será la aproximación de la función tratada, pero se debe tomar en cuenta que si se exagera en el número de neuronas ocultas, se puede

obtener una red inestable o un tiempo de entrenamiento muy lento, especialmente con algoritmos de entrenamiento cuyos cálculos son de alta complejidad como el algoritmo de Levenberg-Marquardt [3, 8, 13].

Después de una serie de pruebas se llegó al diseño más eficiente de la red, la cual presenta una buena aproximación de la transformada de distancia con la que se entrena, así como un tiempo de entrenamiento rápido sin necesidad de modificar los valores de las coordenadas y sus distancias que son los *patterns*, es decir, a la entrada de la red se presenta el par de coordenadas tal cual, y a la salida se espera su respectiva distancia. El perceptrón multicapa usado se define como una red de 2 entradas, dos capas ocultas, la primera de nueve neuronas y la segunda de 25, y una neurona de salida, es decir, 2x9x25x1. Las dos entradas corresponden al par (x, y) que son enteros dentro del rango $1 \leq x,y \leq 64$. Las dos capas ocultas sirven para aproximar mejor la función, y para hacer el entrenamiento más rápido se escogió la *hyperbolic tangent sigmoid* como su función de activación. En la primera capa oculta se eligieron 9 neuronas como representación de las nueve localidades en una matriz que se toman en cuenta al juntar los operadores expuestos en el Capítulo 2 para calcular la transformada de distancia de una imagen; las 9 neuronas de la primera capa oculta están directamente relacionadas con las entradas (x, y) al igual que lo están las distancias dentro del cuadro alrededor de cada elemento (x, y) incluyéndose este mismo. Debe recordarse que para el cálculo de la transformada de distancia, sólo se fueron tomando en cuenta las características (valores) de los 9 elementos incluyéndose el elemento para el que se calcula la distancia de sus vecinos adyacentes, y es lo que se busca con las 9 neuronas de la primera capa oculta. En la segunda capa oculta se hicieron cambios en el número de neuronas a partir de 18, que es el doble de neuronas de la primera capa oculta,

buscando la configuración que mejor extrajera las características de las entradas en conjunto para hacer una buena aproximación de la función en cuestión. La capa de salida se conforma sólo de una neurona que es la encargada de obtener la distancia del píxel localizado en (x, y) cuyo valor es real y positivo, y puede estar dentro del rango $0 \leq x,y \leq 64$, por lo que su función de activación es *linear* que nos puede dar cualquier valor real. La Figura 4.3 muestra la configuración del perceptrón diseñado y la forma de las funciones de activación de cada capa.

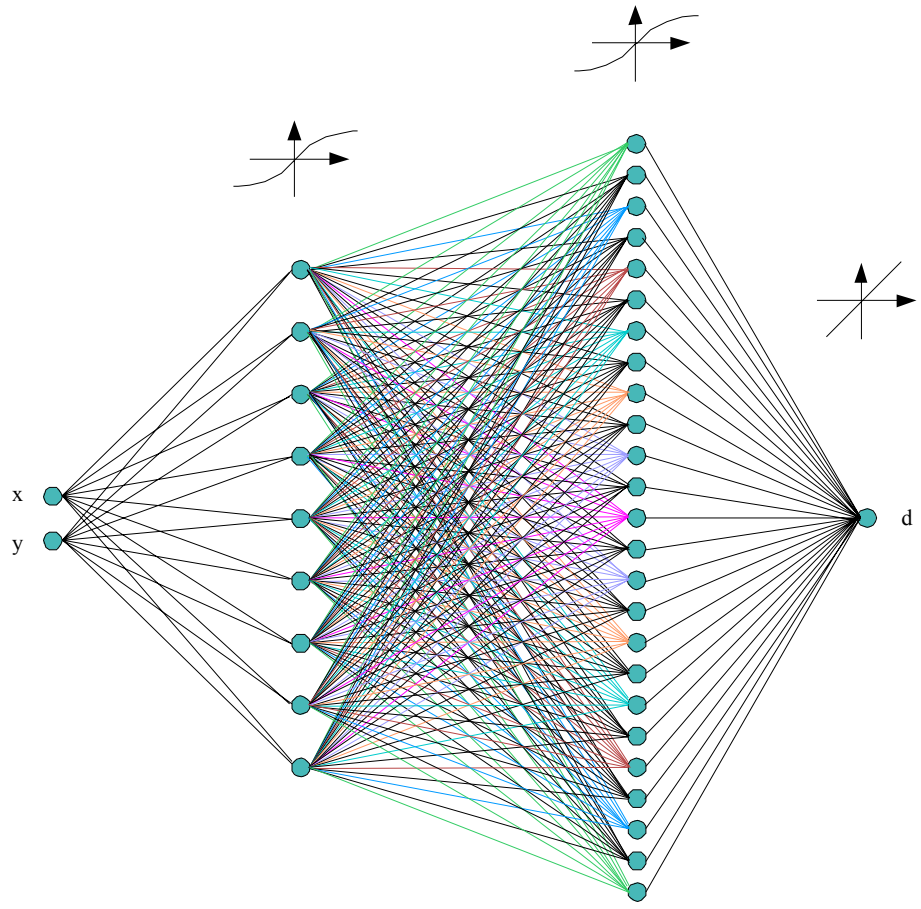


Figura 4.3. Perceptrón multicapa de 2x9x25x1 completamente conectado. Las capas ocultas tienen función de activación *hyperbolic tangent sigmoid* y la capa de salida *linear*.

En el Apéndice A se describe el programa empleado para entrenar la red neuronal usada, donde se supone que ya se tiene una matriz con la transformada de distancia de la imagen que se quiere "aprender" y sólo se aproxima a un MSE de 0.05 para los 1922 *patterns* tomados de los posibles 4096 de una transformada de distancia de 64x64. La Figura 4.4 muestra el resultado que se obtiene del entrenar el MLP descrito anteriormente para "aprender" la transformada de distancia de la imagen de la Figura 4.2(a).

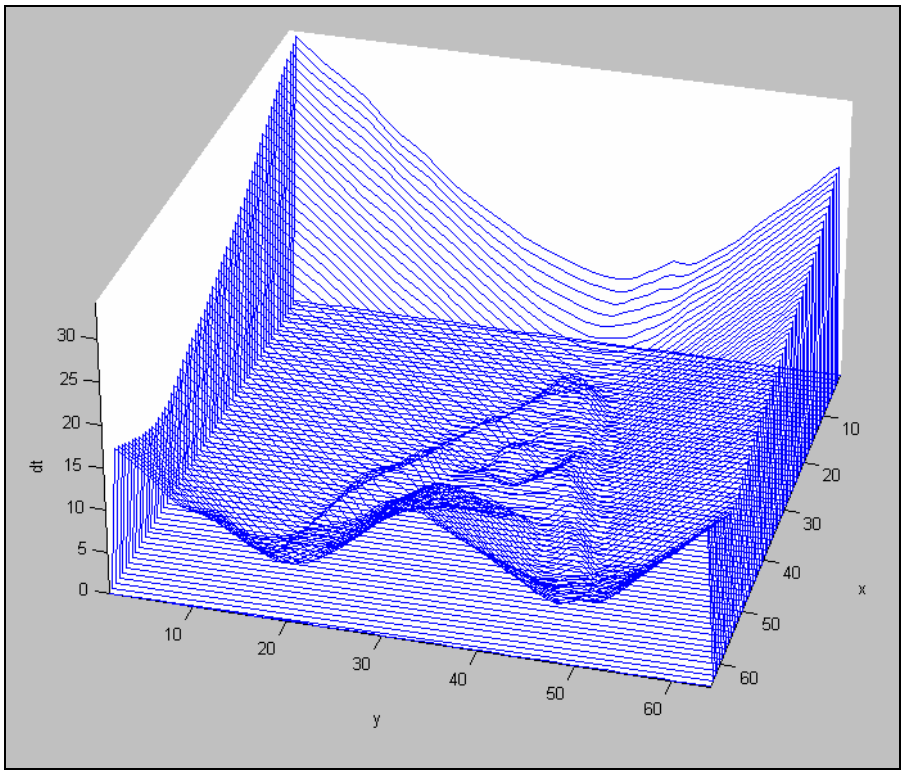


Figura 4.4. Transformada de distancia calculada por el MLP propuesto.

Al comparar las Figuras 4.2(c) y 4.4, se ve que el MLP propuesto si calcula una muy buena aproximación de la transformada de distancia aunque no la generaliza perfectamente como se puede apreciar en la Figura 4.5 que muestra una porción de la transformada de distancia de una imagen de 64x64 comparada con su aproximación..

8.5	8.1	7.1	6.2	5.2	4.2	3.8	2.8
8.1	7.1	6.7	5.7	4.8	3.8	2.8	2.4
7.7	6.7	5.7	5.2	4.2	3.4	2.4	1.4
7.1	6.2	5.2	4.2	3.8	2.8	2.0	1.0
6.7	5.7	4.8	3.8	2.8	2.4	1.4	1.0
6.2	5.2	4.2	3.4	2.4	1.4	1.0	0.0
5.7	4.8	3.8	2.8	2.0	1.0	0.0	1.0
5.2	4.2	3.4	2.4	1.4	1.0	0.0	1.0

(a)

8.7	7.9	7.0	6.1	5.2	4.3	3.4	2.5
8.1	7.3	6.4	5.5	4.6	3.7	2.9	2.0
7.6	6.7	5.8	5.0	4.1	3.2	2.3	1.5
7.1	6.2	5.3	4.4	3.5	2.6	1.8	1.1
6.7	5.8	4.8	3.9	2.9	2.0	1.3	0.7
6.3	5.3	4.3	3.4	2.4	1.5	0.8	0.5
5.9	4.9	3.9	2.9	1.9	1.0	0.4	0.5
5.4	4.5	3.5	2.4	1.4	0.6	0.3	0.8

(b)

Figura 4.5. (a) Transformada de distancia. (b) Aproximación de la Transformada de distancia.

La Figura 4.5 muestra que existe aún error en algunas distancias que por lo general son las que no se tomaron como *patterns* para el entrenamiento de la red. No obstante, esta aproximación es efectiva para el reconocimiento. El uso de más decimales retarda el entrenamiento ya que se busca un menor MSE, y sin embargo, aunque se aproxima mejor la transformada de distancia, al hacer pruebas se obtienen los mismos resultados en el reconocimiento.

4.2.3 Leer Imagen y Reconocerla

Ya que se tienen las redes entrenadas, cada una con una imagen distinta, se procede a probar el desempeño del reconocimiento, el cual se observa al presentar una imagen desconocida a las redes entrenadas y asociarla con la que más tenga parecido. La imagen

debe ser *grayscale* de 64x64. La imagen leída debe ser procesada para poder reconocerla ya que las redes están entrenadas con imágenes binarias que contienen sólo el contorno de interés.

El reconocimiento por medio de la transformada de distancia es en si sencillo ya que obedece sólo a calcular las distancias que existen entre los píxeles que están en la misma ubicación dentro de dos imágenes distintas. Es común tener una base de datos con las transformadas de distancias de imágenes "conocidas", en nuestro caso las redes neuronales entrenadas, y usarlas para compararlas con una imagen desconocida de la cual no es necesario sacar su transformada ya que sólo se toman en cuenta los píxeles que conforman el contorno. Como los píxeles que conforman el contorno tienen un valor de 0, entonces se toma el valor absoluto de la diferencia que hay entre éstos y aquellos con mismas coordenadas en las transformadas de las imágenes que se tienen en la base de datos, que a fin de cuentas resulta ser el valor que tienen los píxeles de las trasformadas, puesto que el valor absoluto de la diferencia 0-a es el valor absoluto de a. Aquí como las transformadas de distancia tienen únicamente valores positivos, entonces ya se cuenta con los valores absolutos.

En el Apéndice A se describe el programa usado para efectuar el reconocimiento de imágenes que no fueron usadas para el entrenamiento de los MLPs.