Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias Físico Matemáticas

Reconocimiento de patrones en imágenes mediante Aprendizaje Profundo.

Tesis presentada al

Colegio de Física

Como requisito parcial para obtener el grado de

Licenciado en Física

por

Pérez Lima Dalí del Ángel

Asesorado por

Dr. Juan Castillo Mixcóatl

Puebla Pue.

28 de junio del 2021

Reconocimiento de patrones en imágenes mediante Aprendizaje Profundo.

.

Dalí del Ángel Pérez Lima

Dr. Juan Castillo Mixcóatl

B U A P

Título: Reconocimiento de patrones en imágenes mediante Aprendizaje Profundo.

Estudiante: Dalí del Ángel Pérez Lima

Comité:

_____

Dr. Severino Muñoz Aguirre
Presidente.

_____

Dra. Georgina Beltrán Pérez
Secretaria.

_____

Dra. Rosibel Carrada Legaria
Vocal.

_____

Dr. Juan Castillo Mixcóatl
Asesor

_____

Dr. Carlos Ignacio Robledo Sánchez
Suplente

CONTENIDO

Resumen

Actualmente existen distintas actividades que no puede ser analizadas o desempeñadas por una computadora mediante técnicas tradicionales del conocimiento como leyes científicas o el uso de ecuaciones diferenciales. Ejemplos de tales actividades pueden ser el reconocimiento de patrones en imágenes, reconocimiento de lenguaje, reconocimiento de voz, ente otras. En todas estas actividades existe una constante: resulta ser sumamente complicado implementarlas con la programación tradicional con el ya mencionado uso de leyes de la ciencia o de algún conjunto de ecuaciones diferenciales que intenten analizar este tipo de fenómenos. Sin embargo, casi cualquier ser humano puede desempeñar cualquiera de estas tareas prácticamente sin siquiera pensarlo. Por esta razón, en los últimos años ha habido una enorme investigación para desarrollar técnicas que imiten el funcionamiento del cerebro humano para llevar a cabo este tipo de actividades. En este trabajo se presentan las ideas fundamentales acerca del Aprendizaje Profundo, Aprendizaje de Maquina y redes neuronales. Estos conceptos básicos permitan entender y conocer el funcionamiento de estas técnicas y la relación que existe entre ellas. Se muestra también un ejemplo simple de reconocimiento de patrones en una imagen mediante el uso de una red neuronal de convolución. Creemos que este trabajo será útil para aquellas personas que se interesen en estos temas y se adentren en ellos por primera vez.

Palabras clave:

Inteligencia artificial, Aprendizaje de Maquina, Aprendizaje Profundo, Reconocimiento de patrones, redes neuronales

Introducción

Como se ha mencionado en el resumen, existen tareas en las cuales la programación tradicional no ofrece resultados satisfactorios y el nivel de complejidad puede ser abrumador, la introducción de las redes neuronales es una respuesta simple y fácil de implementar que hoy en día ha dado resultados espectaculares. En este trabajo se pretende mostrar las ideas básicas detrás de los conceptos de Aprendizaje de Máquina, el Aprendizaje Profundo y las Redes Neuronales. A continuación, se describen los objetivos de este trabajo es el siguiente.

Objetivo principal

Estudiar y comprender el funcionamiento básico de las Redes Neuronales en la implementación del Aprendizaje de Maquina y el Aprendizaje Profundo.

Objetivos particulares

Para lograr el objetivo principal se proponen los siguientes objetivos particulares:

1. Implementación de programas para una Red Neuronal Monocapa.
2. Implementación de programas para una red con capas ocultas.
3. Ejemplo de aplicación de los conceptos de aprendizaje profundo utilizando el algoritmo de propagación inversa.
4. Ejemplo de aplicación de una Red Neuronal Convolucional para la clasificación de información en imágenes.

Capítulo 1. Aprendizaje de Máquina

En este capítulo se presentan los fundamentos necesarios para entender los conceptos relacionados con el Aprendizaje de Máquina (en inglés, Machine Learning), además se presentan los problemas que surgen en esta técnica, se muestran también las variantes del Aprendizaje de Máquina y algunas de sus aplicaciones.

El concepto de inteligencia se puede definir como la habilidad de procesar información para tomar decisiones futuras [1]. El campo de la inteligencia artificial es el encargado de la formación de algoritmos artificiales para la resolución de problemas. El Aprendizaje de Máquina es una rama de la inteligencia artificial encargada de la formación de este algoritmo sin la necesidad de un programador. Existen distintas definiciones de lo que se considera el Aprendizaje de Máquina, por ejemplo:

- *"El aprendizaje de maquina es la ciencia de programar computadoras de manera que ellas puedan aprender a partir de una serie de datos"*
- *"El aprendizaje de Máquina es el campo de estudio en el que se le da a la computadora, la habilidad de aprender sin que esta sea programada explícitamente"*
- *"Un programa de computadora se dice que aprende de la experiencia E con respecto de alguna tarea T y alguna medida del desempeño P, si su desempeño en T, medido por P, mejora con la experiencia E" [4].*

Todas estas definiciones ahora mismo pueden resultar confusas, en el sentido de que no es claro cómo realizar el proceso descrito por ejemplo en la tercera definición (la cual intenta ser más concreta). Como se verá más adelante, este algoritmo de inteligencia artificial en realidad será reducido a operaciones tan sencillas como multiplicaciones (matriciales) y la determinación del error en la salida ofrecida por el algoritmo el cual es utilizado para corregir los términos multiplicativos (matrices) que serán utilizados para evaluar una siguiente salida que puede nuevamente repetir el proceso anterior, generando un proceso equivalente a lo que concebimos como aprendizaje (minimización del error).

El proceso que emplea el Aprendizaje de Máquina es identificar características de los datos para después crear un modelo con estos, entre más características se quieran identificar el modelo se hará más complicado y con esto surgen algunos problemas que posteriormente se solucionan con el Aprendizaje Profundo (en inglés, Deep Learning), este es una rama del Aprendizaje de Máquina, la Fig. 1.1 representa la relación entre estos:

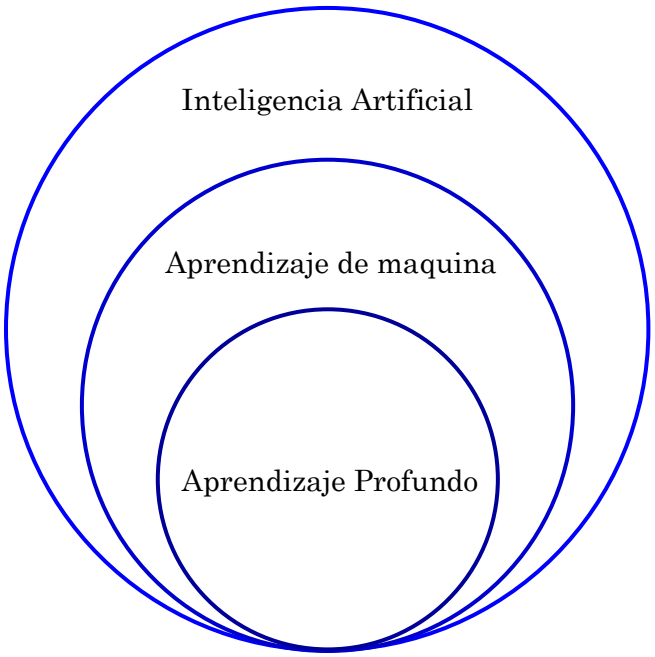

Fig. 1.1 Relación entre inteligencia artificial, Aprendizaje de Mquina y aAprendizaje Profundo.

El Aprendizaje de Máquina es una técnica de análisis que da como resultado un **modelo**, basado en datos de entrenamiento y es similar a nuestro proceso de aprendizaje. En la Fig. 1.2 se muestra un esquema de la relación entre estos conceptos.

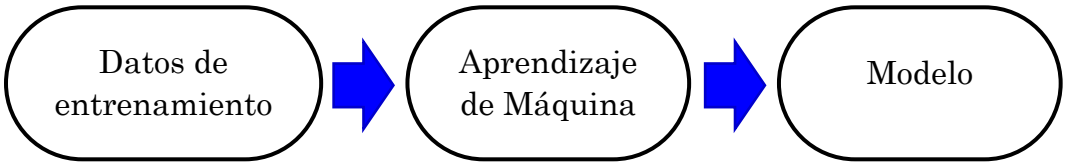

Fig. 1.2. Pasos comúnes en el proceso de Aprendizaje de Máquina.

El Aprendizaje de Máquina fue creado para solucionar problemas difíciles de resolver para una computadora. Este tipo de problemas son aquellos que involucran el uso de conocimiento e inteligencia propios de los seres humanos, por ejemplo, el reconocimiento de patrones en imágenes, el reconocimiento del habla y el procesamiento natural del lenguaje. En el caso del Aprendizaje de Máquina, el modelo es todo aquello que deseamos hacer por ejemplo identificar rostros en una imagen, filtrar correo electrónico importante del "spam", etc. En algunas ocasiones a este modelo se le conoce también como **hipótesis**. El Aprendizaje de Máquina utiliza el modelo obtenido anteriormente para analizar un conjunto de datos de entrada o de interés y entrega una salida o resultado de estos datos, este proceso se muestra en la Fig. 1.3 [1, 3].

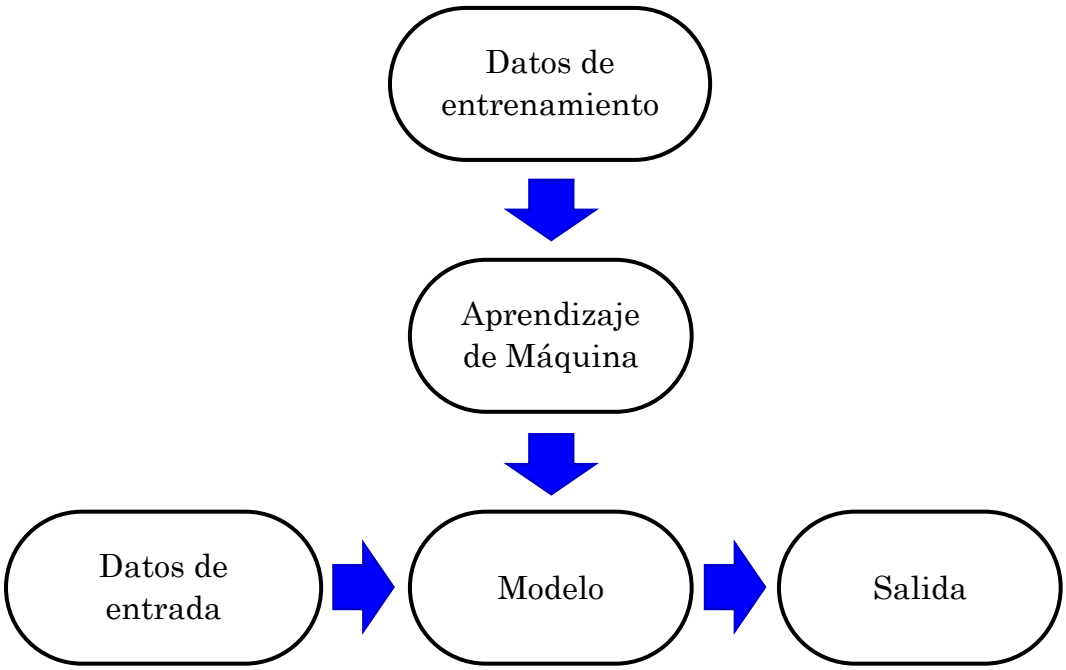

Fig. 1.3. Pasos usuales en el proceso de la generación del modelo en el Aprendizaje de Máquina.

La manera en cómo garantizamos que el modelo encontrado realice las acciones deseadas tanto con los datos de entrenamiento como con los datos reales se le conoce como **generalización.** La eficacia en el procesamiento deseado en el modelo depende fuertemente de que tan bien ha sido implementado el proceso de generalización. Existen usualmente algunos problemas típicos de este proceso. A continuación, vamos a definir algunos de los más comunes.

1.1 PROBLEMAS EN EL APRENDIZAJE DE MÁQUINA

1.1.1 Sobreajuste

Uno de los principales factores que reduce el rendimiento de la generalización es el **sobreajuste**, éste ocurre cuando hay una gran diferencia ente los valores de los datos de entrenamiento y los datos de entrada. Por ejemplo, supongamos que estamos interesados en separar distintas clases de objetos. Digamos entonces, que en los datos de entrenamiento tenemos las variables ○ y ●, y deseamos dibujar una línea que separe los dos grupos en la imagen, uno con la variable ○ y otro con la variable ●. La Fig. 1.4A) y Fig. 1.4B) muestran un conjunto de datos de entrenamiento y una posible salida correcta, respectivamente.

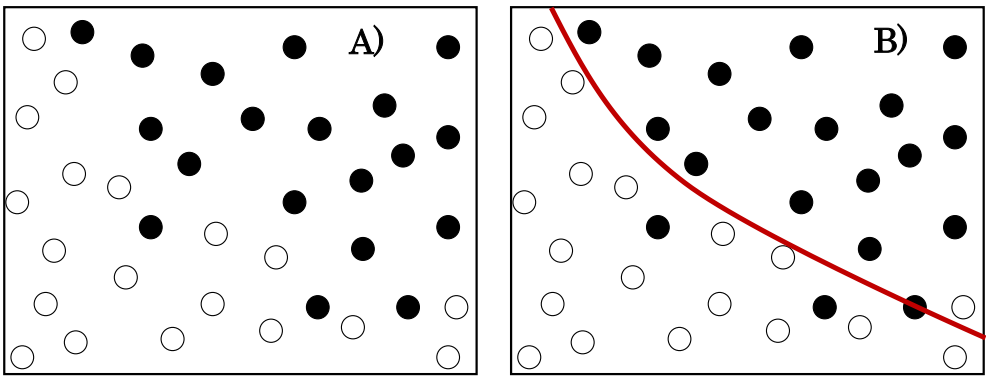

Fig. 1.4. A) Datos de entrada. B) Datos de salida con la línea que separa correctamente a los datos en dos grupos distintos.

Como puede verse en la Fig. 1.4B, a pesar de que existen algunos elementos que se internan más allá de esta línea, en general podemos decir que esta agrupación es aceptable. Sin embargo, podríamos estar tentados a "mejorar" nuestro modelo. Por lo que este nuevo modelo daría ahora un resultado como el que se muestra en la Fig. 1.5.

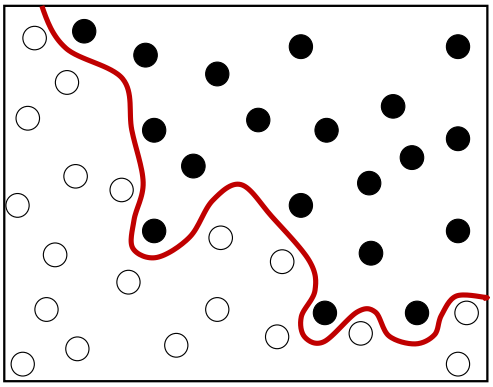

Fig. 1.5. Un modelo "mas" preciso sería capaz de agrupar los datos de mejor manera, tal y como de ilustra en este esquema.

En este caso uno pude pensar que el modelo realiza un excelente trabajo, pronto veremos que esto no necesariamente es cierto.

Supongamos ahora, que llega el conjunto de elementos de entrada de la Fig. 1.6 Es claro, que el dato ● no corresponde a ninguno de los dos grupos, entonces uno puede preguntarse ¿a cuál de estos dos grupos pertenece el nuevo elemento?

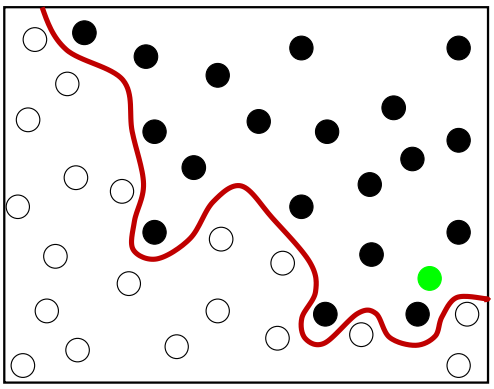

Fig. 1.6. El sobreajuste se genera cuando existe una gran diferencia entre los datos de entrenamiento y los datos a procesar. En este caso, el resultado es una agrupación dudosa de las variables actuales.

Como puede verse de la Fig. 1.6, el modelo clasifica a este nuevo valor en uno de los grupos (el que parezca más adecuado), en este caso, el modelo clasifica este nuevo valor en el grupo de los ●. Sin embargo, uno puede pensar que esto no necesariamente es cierto, por lo que incluso podriamos decir que pertenece al otro grupo.

¿Por que sucede esto? Esto se debe a que los datos de entrada reales, pueden contenter informacion que no es consistente con los datos de entrenamiento, a esta se les puede considerar como ruido. El problema radica, en que el Aprendizaje de Máquina no tiene forma de distinguir esto, ya que este considera todos los datos, incluso el ruido y termina produciendo un modelo inadecuado. Al proceso de forzar al modelo para que trabaje solo con los datos de entrenamiento se le conoce como sobreajuste.

A continuacion se muestran algunas soluciones para resolver el problema del sobreajuste, estas son: la regularización, validación y validación cruzada.

1.1.2 Solución al sobreajuste: Regularización, validación y validación cruzada

Para entender cómo puede resolverse el sobreajuste, sigamos con el problema mencionado anteriormente: deseamos separar los elementos en grupos en una imagen. La **regularización** [1, 4] es un método numérico que intenta hacer del modelo una estructura lo más simple posible y con esto se puede resolver el sobreajuste. Particularmente, en nuestro problema la regularización significa una ecuación polinómica para separar los elementos en la imagen. Por lo tanto, el modelo trazara una curva más sencilla tal y como se observa en la Fig. 1.7. Se puede ver que, bajo esta circunstancia, el modelo no clasifica los datos de manera precisa, pero muestra las características generales de los datos.

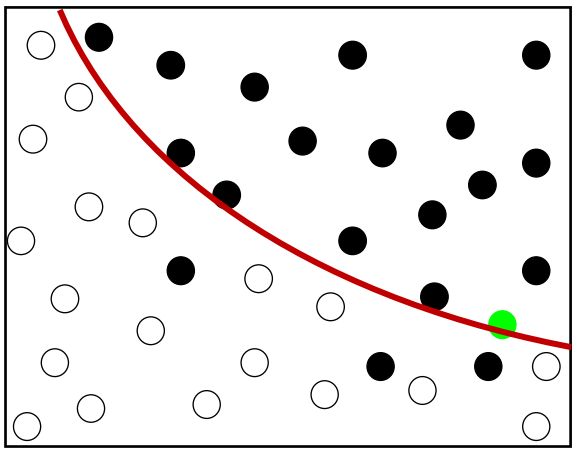

Fig. 1.7. El método de regularización aplicado al problema de agrupamiento de la Fig. 1.6.

De esta manera, a pesar de que no se agrupan de manera precisa si podemos establecer las características generales de los datos y esto en muchas circunstancias, puede ser un resultado suficiente.

Otra forma de resolver el problema de sobreajuste es mediante el proceso de **validación**, esta técnica divide los datos de entrenamiento en dos conjuntos, uno para entrenar al modelo y el otro para generar un proceso de validación. Como regla general, usualmente se utiliza una relación 8:2 (Datos de entrenamiento/Datos para validación). De esta manera primero se entrena el modelo con el primer grupo y enseguida se ocupa el segundo grupo para comprobar si el modelo esta sobre ajustado, si es así el modelo se modifica y se repite el proceso una vez más, hasta tener un buen rendimiento, la división de los datos de entrenamiento se muestra en la Fig. 1.8

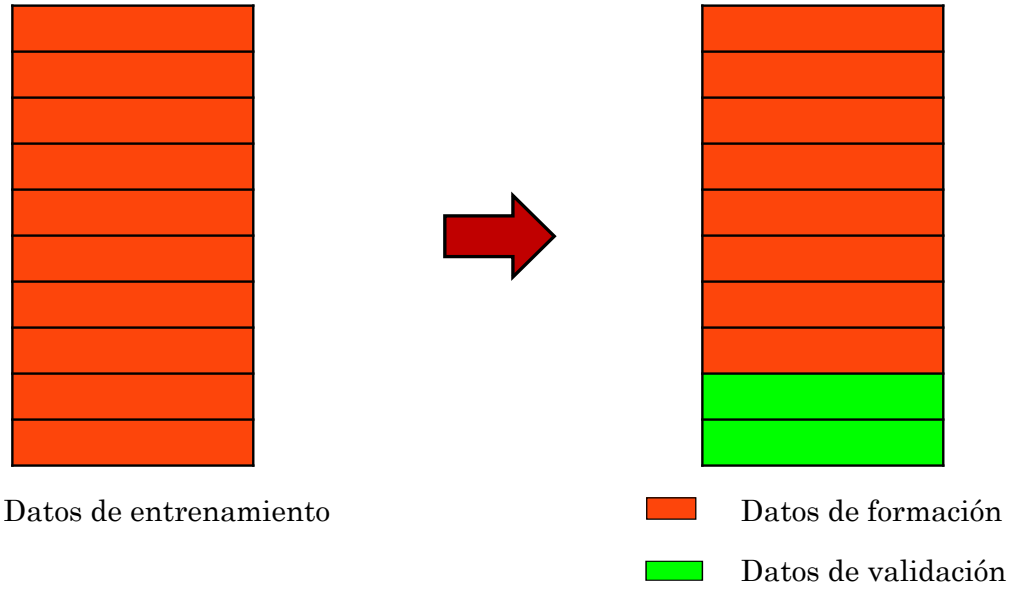

Fig. 1.8. En el proceso de validación el conjunto de datos de entrenamiento se divide en dos partes. Uno para generar el entrenamiento y otro para realizar un proceso de validación del modelo. Usualmente esto se da con una relación 8:2.

Una variante del proceso de validación es lo que se conoce como validación cruzada. Este es un proceso similar al anterior, la diferencia radica en que el conjunto de datos de validación se elige aleatoriamente. La razón de esto es reducir aún más el sobreajuste ya que este se puede presentar aun en el conjunto de entrenamiento, y con la aleatoriedad se minimiza este sobre ajuste. La división de los datos se muestra en la Fig. 1.9

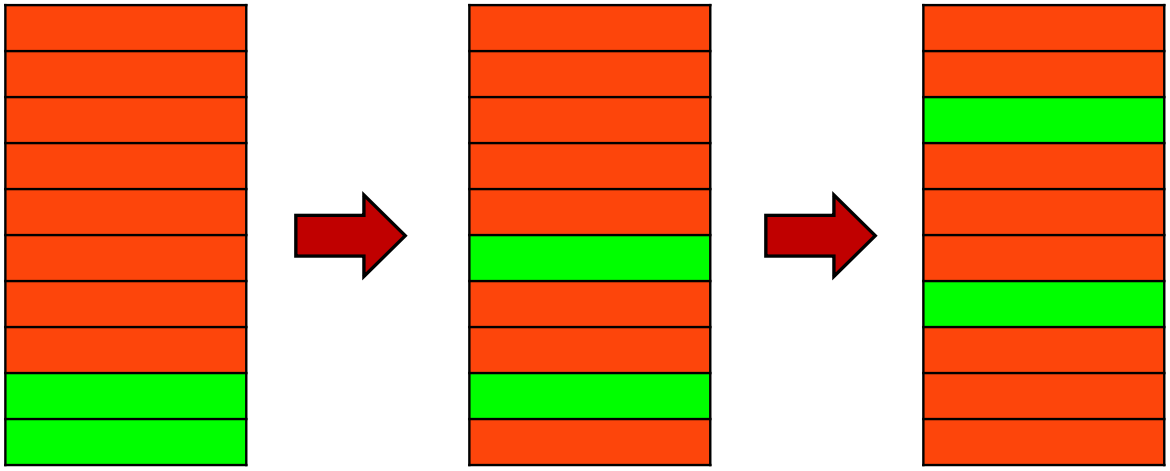

Fig. 1.9. El proceso de validación cruzada divide aleatoriamente los datos de entrada en los conjuntos de validación y de entrenamiento, esto con el fin de reducir el sobreajuste.

A continuación, vamos a describir las clases típicas de Aprendizaje de Máquina.

1.2 TIPOS DE APRENDIZAJE DE MÁQUINA

Existen tres tipos de Aprendizaje de Máquina: Aprendizaje Supervisado, Aprendizaje no Supervisado y Aprendizaje Reforzado [1, 3].

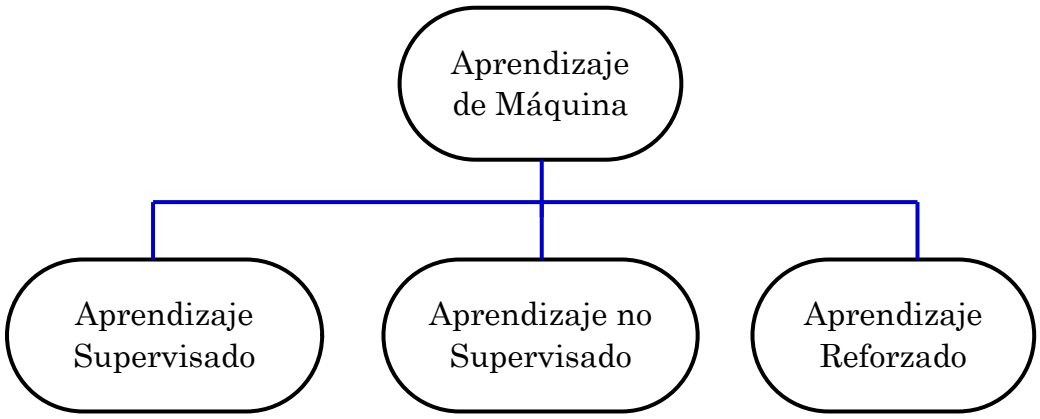

Fig. 1.10. Tipos de Aprendizaje de Máquina.

El conjunto de los datos de entrenamiento se puede expresar como:

$$D = \{(x_i, y_i)\}_{i=1}^{N} \tag{1.1}$$

Donde x_i, y_i son los i-ésimos datos de entrada y salida de la red, respectivamente. N es el número total de datos. Los datos de entrenamiento varían

dependiendo del tipo de aprendizaje que se utiliza, a continuación, veamos la descripción de cada uno de estos tipos.

1.2.1 Aprendizaje Supervisado

Los datos de entrenamiento para el Aprendizaje Supervisado requieren dos argumentos: el dato de entrada y la salida correcta. Esta técnica modifica el modelo de tal forma que la diferencia entre la salida real y la salida correcta sea mínima.

Una de las aplicaciones más comunes del Aprendizaje Supervisado es la clasificación. Esta se centra en encontrar el grupo al que pertenecen los datos. Los datos de entrenamiento para esta aplicación requieren dos argumentos el dato de entrada y el grupo o clase al que pertenece.

Una aplicación de la clasificación es el reconocimiento de dígitos escritos a mano. Por ejemplo, pueden usarse matrices de 5x5 en las que las entradas de las matrices pueden ser 1 o 0. Si el 0 lo representaremos por el color blanco y el 1 por el color verde, el conjunto de los datos de entrenamiento D luciría como lo que se muestra en la Fig. 1.11. En este caso, las matrices son los datos de entrada y los números son los grupos a los que pertenecen estos datos de entrada.

$$D=\left\{\left(\begin{pmatrix}0&0&1&0&0\\0&1&1&0&0\\0&0&1&0&0\\0&0&1&0&0\\0&1&1&1&0\end{pmatrix},1\right),\left(\begin{pmatrix}0&1&1&1&0\\0&0&0&1&0\\0&1&1&1&0\\0&1&0&0&0\\0&1&1&1&0\end{pmatrix},2\right),\left(\begin{pmatrix}0&1&1&1&0\\0&0&0&1&0\\0&1&1&1&0\\0&0&0&1&0\\0&1&1&1&0\end{pmatrix},3\right),\left(\begin{pmatrix}0&1&0&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&1&0\\0&0&0&1&0\end{pmatrix},4\right),\left(\begin{pmatrix}0&1&1&1&0\\0&1&0&0&0\\0&1&1&1&0\\0&0&0&1&0\\0&1&1&1&0\end{pmatrix},5\right)\right\}$$

Fig. 1.11. Datos de entrenamiento para el Aprendizaje Supervisado en el que se presentan los datos de entrada y salida correcta.

Otra aplicación típica para el Aprendizaje Supervisado es la regresión. esta utiliza como dato de salida correcta valores y con esta se estima la tendencia de los datos reales.

Para ejemplificar la regresión consideremos que conociendo las temperaturas de una cierta región medidas con intervalos de tiempo constantes de una hora durante todo un día, es necesario crear un modelo matemático para predecir las temperaturas faltantes de ese día, además de corregir las mediciones en caso de que una de estas sea incorrecta. La Fig. 1.12 muestra los datos de entrada para este ejemplo. En esta figura, el primer valor de los pares ordenados es la hora y el segundo la temperatura medida en esa hora. la Fig. 1.13 muestra la gráfica de temperatura vs hora y la regresión de los datos.

$$D=\left\{\begin{array}{c}(00{:}00,9°C),(01{:}00,8°C),(02{:}00,7°C),(03{:}00,7°C),(04{:}00,7°C),(05{:}00,8°C),\\(06{:}00,9°C),(07{:}00,21°C),(08{:}00,12°C),(09{:}00,13°C),(10{:}00,15°C),(11{:}00,18°C),\\(12{:}00,21°C),(13{:}00,23°C),(14{:}00,24°C),(15{:}00,23°C),(16{:}00,22°C),(17{:}00,20°C),\\(18{:}00,19°C),(19{:}00,18°C),(20{:}00,16°C),(21{:}00,13°C),(22{:}00,11°C)(23{:}00,10°C)\end{array}\right\}$$

Fig. 1.12. Los datos de entrenamiento para la regresión contienen números reales. Los datos de este ejemplo son las temperaturas tomadas durante todo un día, el primer dato es la hora y el segundo es la temperatura medida en esa hora.

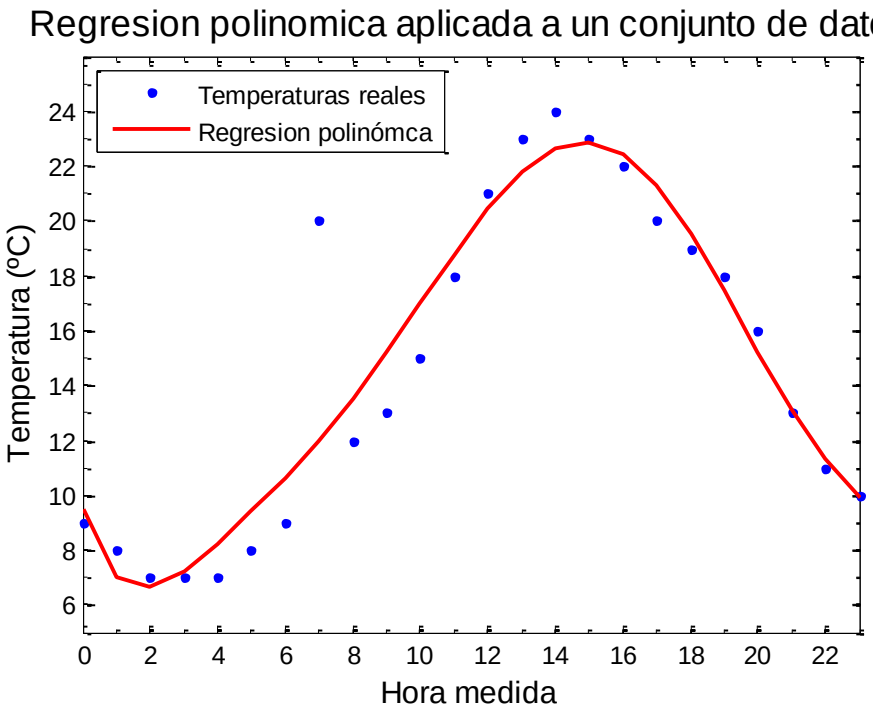

Fig. 1.13. El proceso de regresión aproxima los datos de salida reales (temperatura) a una curva polinómica, reduciendo así los posibles errores que surgen al hacer las mediciones de la temperatura.

1.2.2 Aprendizaje no Supervisado

El Aprendizaje no Supervisado solo contiene el dato de entrada y se utiliza generalmente para investigar las características generales de los datos y pre procesarlos. Este aprendizaje "descubre" estructuras interesantes dentro de los datos. Una aplicación de este es la agrupación (en inglés clustering), el objetivo es encontrar diferentes grupos dentro del conjunto de los datos de entrada, este permite variar el número de agrupaciones en función del tamaño del problema, permitiendo al usuario el control del grado de similitud entre miembros de las mismas agrupaciones en términos de una constante definida por el usuario llamada el parámetro de vigilancia.

1.2.3 Aprendizaje Reforzado

El Aprendizaje Reforzado emplea un conjunto de entradas, alguna salida y la calificación según los datos de entrenamiento [1, 4]. Este tipo de aprendizaje es muy útil cuando es necesario un proceso de optimización. Comúnmente se utiliza en el área de control e incluso en videojuegos.

{Entrada, alguna respuesta, grado para esta salida}

En este trabajo se hablará de aquí en adelante únicamente del Aprendizaje Supervisado debido a que es una de las técnicas más comunes en el Aprendizaje de Máquina. A continuación, se muestran los conceptos relacionados con las redes neuronales que son la manera más común de implementar el Aprendizaje de Máquina.

Capítulo 2. Redes Neuronales

En el capítulo anterior se presentaron los fundamentos del Aprendizaje de Máquina, en este se presenta la estructura fundamental de una Red Neuronal. Se le da prioridad al Aprendizaje Supervisado ya que este es el más utilizado para fines prácticos, se presentarán las variantes de las redes dependiendo de qué tan complicada sean, además se introduce el concepto de regla de aprendizaje utilizado para modificar y entrenar a una Red Neuronal.

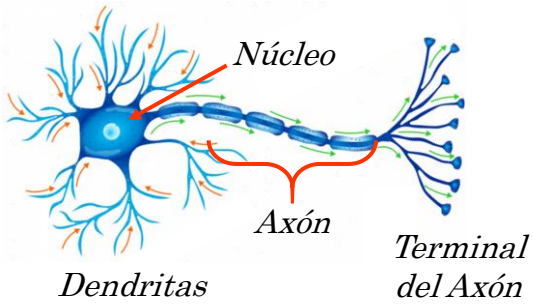

Fig. 2.1. Una Red Neuronal está formada por una gran cantidad de nodos conectados unos con otros, la conexión y comunicación entre estos es similar al proceso que hay entre neuronas reales en el cerebro.

Una Red Neuronal está formada por una gran cantidad de nodos conectados unos con otros, la conexión y comunicación entre estos es similar al proceso que hay entre neuronas reales en el cerebro [1, 3, 4]. Se puede hacer una analogía entre los nodos y las neuronas reales. La información de la Red Neuronal se almacena en forma de pesos y sesgos. La Fig. 2.2 representa a un solo nodo con tres señales de entrada.

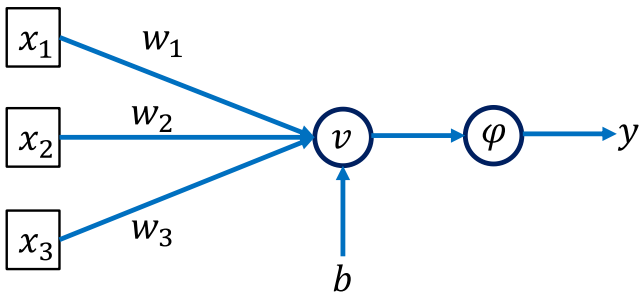

Fig. 2.2. Nodo de una red neuronal donde X_i son las señales de entrada, W_i son los pesos de las entradas, b es el sesgo, v es la suma ponderada de las entradas y la activación de v es la salida del nodo Y.

Los pesos W_i determinan la importancia de la información de entrada, la suma ponderada (v) es la sumatoria del producto de la señal de entrada por su respectivo peso más el valor del sesgo, en forma general se puede expresar de la siguiente forma:

$$v = \sum_{i=1}^{N} x_i w_i + b \qquad (2.1)$$

La ecuación (2.1)puede también ser escrita de forma matricial como:

$$v = WX + b \tag{2.2}$$

donde:

$$W = [w_1 \quad \dots \quad w_N] \tag{2.2a}$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \tag{2.2b}$$

La Ecuación (2.2a) es el vector fila de pesos y la Ecuación (2.2b) es el vector columna de entradas, estas ecuaciones son para N nodos de entrada y un solo nodo de salida.

La función de activación φ, determina el comportamiento del nodo y produce la salida de este, y. Entonces la función de activación se aplica a la suma ponderada tal y como se muestra en la Ecuación (2.3).

$$y = \varphi(v) \tag{2.3}$$

En el futuro representaremos a un nodo de forma más reducida como se muestra en la Fig. 2.3 y omitiremos el factor de sesgo.

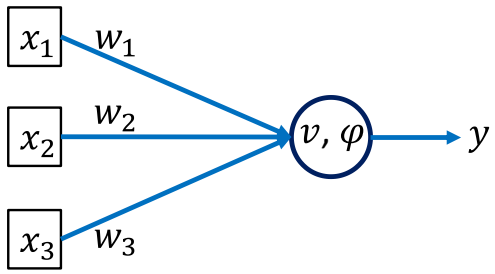

Fig. 2.3. Nodo de una Red Neuronal donde X_i son las señales de entrada, W_i los pesos entre las entradas y la salida Y.

2.1 CAPAS DE UNA RED NEURONAL

Una Red Neuronal está formada por diversas capas de nodos conectados unos con otros como se muestra en la Fig. 2.4

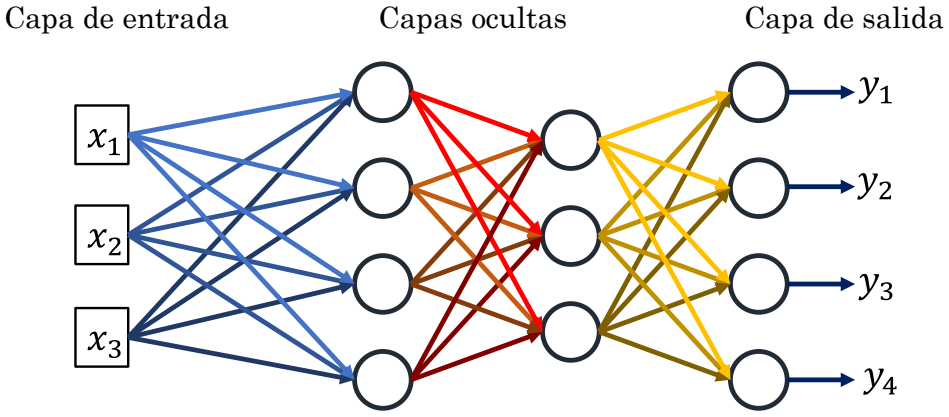

Fig. 2.4. Una Red Neuronal está formada por diversas capas: la capa de entrada, las capas ocultas y la capa de salida.

La capa de entrada está formada por todos los nodos de entrada, esa capa solo se encarga de transmitir información a las siguientes capas. La capa de salida es el resultado final de la Red Neuronal y las capas que están entre estas son las capas ocultas, se denominan así porque no son accesibles desde fuera de la Red Neuronal. La señal avanza de izquierda a derecha capa por capa, desde la capa de entrada hasta llegar a la capa de salida.

Una Red Neuronal se clasifica por el número de capas ocultas que tiene, si no tiene capas ocultas se llama Red Neuronal Monocapa, si tiene solo una capa oculta se llama Red Neuronal Superficial (Shallow Neural Network, en inglés) y si tiene dos o más capas ocultas se denomina Red Neuronal Profunda (Deep Neural Network, en inglés).

La Ecuación (2.2) define la suma ponderada para un solo nodo, esta se puede generalizar para una capa de M nodos como se muestra en la Fig. 2.5. La Ecuación (2.4) se ocupa para N entradas y M salidas.

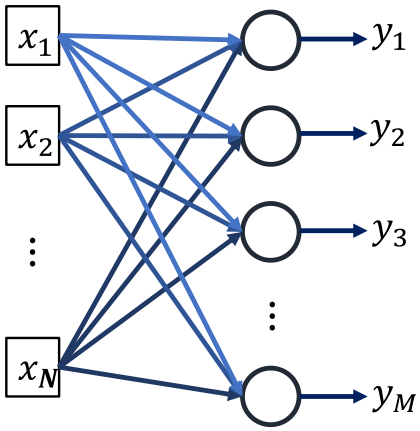

Fig. 2.5. Red Neuronal con N nodos de entrada y M nodos de salida.

$$V = WX + B \tag{2.4}$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \tag{2.4a}$$

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1N} \\ \vdots & & \vdots \\ w_{M1} & \cdots & w_{MN} \end{bmatrix} \tag{2.4b}$$

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} \tag{2.4c}$$

$$V = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix} \tag{2.4d}$$

Donde N es el número de nodos de entrada, M es el número de nodos de salida, B es el vector de sesgo y V es el vector con las sumas ponderadas de cada nodo, los pesos para el i-ésimo nodo se encuentran en la i-ésima fila.

Posteriormente se aplica la función de activación al vector V para obtener el vector de salidas de esta capa, Y.

$$Y = \varphi(V) = \begin{bmatrix} \varphi(v_1) \\ \vdots \\ \varphi(v_M) \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} \tag{2.5}$$

Para una Red Multicapa se realiza el mismo proceso, pero en esta el vector de salida de la primera capa se utiliza como vector de entrada para la segunda capa y este proceso se repite hasta llegar a la capa de salida.

2.2 APRENDIZAJE SUPERVISADO DE UNA RED NEURONAL

La Red Neuronal almacena información en forma de pesos, entonces para entrenar a la Red Neuronal con nueva información hay que cambiar estos pesos, a este proceso de cambiar los pesos con la nueva información se le llama Regla de Aprendizaje.

2.2.1 Regla Delta

Una Regla de Aprendizaje que puede ser utilizada para entrenar una Red Neuronal Mono capa es la llamada Regla Delta, con una función de activación lineal. En esta Regla de Aprendizaje, se ingresan los datos de entrada, se calcula la diferencia entre los datos de salida correctos y los datos de salida reales y a continuación se modifican los pesos de la red, esto se repite hasta que el error es aceptable.

Matemáticamente la Regla Delta, se puede expresar como [1, 3, 4]:

$$w_{ij} \leftarrow w_{ij} + \alpha e_i x_j \tag{2.6}$$

$$e_i = d_i - y_i \tag{2.6a}$$

Con:

x_j es el valor del j-ésimo nodo de entrada.

d_i es la salida correcta del i-ésimo nodo de salida.

y_i es la salida real del i-ésimo nodo de salida.

e_i es la diferencia ente la salida correcta y la salida real del i-ésimo nodo de salida.

w_{ij} es el peso entre el j-ésimo nodo de entrada y el i-ésimo nodo de salida.

α es la razón de aprendizaje, su valor esta entre 0 y 1.

La razón de aprendizaje, α, determina la velocidad con la que el valor del peso cambia, si esta es muy grande el valor del nodo de salida oscila alrededor del valor correcto, pero no converge. Por el contrario, si α pequeña, la red llega a la solución correcta muy lentamente.

Los pasos para aplicar la Regla Delta son los siguientes:

i. *Inicializar los pesos con valores adecuados.*

ii. *Tomar la entrada del primer dato de entrenamiento e introducirlo a la red, después calcular el error entre la salida real y la salida correcta.*

$$e_i = d_i - y_i$$

iii. *Calcular las actualizaciones de peso de acuerdo con la Regla Delta:*

$$\Delta w_{ij} = \alpha e_i x_j$$

iv. *Ajustar los pesos como:*

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

v. *Repetir los pasos 2-4 con todos los datos de entrenamiento.*

vi. *Repetir los pasos 2-5 hasta que el error sea aceptable.*

El modelo se entrena utilizando los mismos datos de entrenamiento debido a que la Regla Delta busca la solución conforme repite el proceso. Esto se repite ya que al entrenar el modelo con los mismos datos puede mejorarse el modelo.

El número de veces que el modelo realiza los pasos 2-5 con todos los datos de entrenamiento se llaman época (epoch en inglés).

2.2.2 Regla Delta Generalizada

La Regla Delta se vuelve obsoleta para Redes Neuronales complejas, por esta razón, ésta se modifica para funciones de activación arbitraras. La Regla Delta Generalizada se expresa entonces como sigue [1, 3, 4]:

$$w_{ij} \leftarrow w_{ij} + \alpha \delta_i x_j \tag{2.7}$$

$$\delta_i = \varphi'(v_i) e_i \tag{2.7a}$$

$$\Delta w_{ij} = \alpha \delta_i x_j \tag{2.7b}$$

Donde:

e_i es el error ente la salida correcta y la salida real del i-ésimo nodo de salida.

v_i es la suma ponderada del i-ésimo nodo.

φ' es la derivada de la Función de Activación.

Una función de activación muy común para las Redes Multicapa es la función Sigmoide, la cual se define como:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

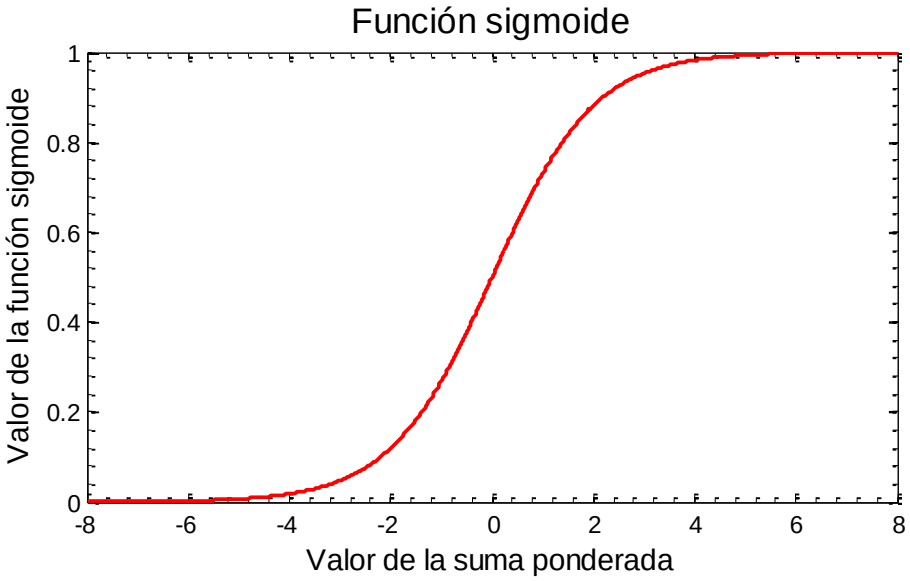

Fig. 2.6. La función Sigmoide se aplica al valor de la suma ponderada y como resultado da un valor entre 0 y 1.

En la Fig. 2.6 se muestra el aspecto de esta función, como se observa, ésta se encuentra acotada entre 0 y 1. Aplicando la Regla Delta Generalizada a esta función obtenemos:

$$\varphi'(x) = \varphi(x)\big(1 - \varphi(x)\big) \tag{2.9}$$

$$\delta_i = \varphi(v_i)\big(1 - \varphi(v_i)\big)e_i \tag{2.9a}$$

$$w_{ij} \leftarrow w_{ij} + \alpha\varphi(v_i)\big(1 - \varphi(v_i)\big)e_i x_j \tag{2.9b}$$

A partir de las Ecuaciones (2.9), puede verse que si la función de Activación $\varphi(x) = x$ entonces, las Ecuaciones (2.7) convergen a las Ecuaciones (2.6), como era de esperarse.

2.2.3 Actualización de pesos

Existen tres métodos típicos para actualizar los pesos de la Red Neuronal que se utiliza para generar el Aprendizaje Supervisado, estos son: el método de Gradiente Descendiente Estocástico, el método de Lotes y el método de Mini Lotes.

En la técnica de Gradiente Descendiente Estocástico (en inglés Stochastic Gradient Descent, SGD) calcula el error para cada dato de entrenamiento e inmediatamente actualiza los pesos, este método es rápido, pero no es muy preciso con

los valores. En este caso se realizan N actualizaciones de pesos en una epoca. Este método se representa gráficamente en la Fig. 2.7.

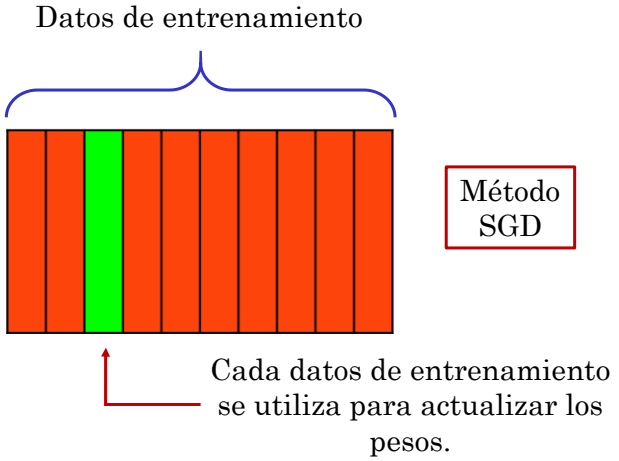

Fig. 2.7. El método de Gradiente Descendiente Estocástico calcula el error de la salida para cada dato de entrenamiento y con cada dato actualiza los pesos de la red.

El método por lotes ingresa cada uno de los datos de entrenamiento y calcula la actualización de pesos para cada uno, después calcula el promedio de todas las actualizaciones de pesos y en seguida ajusta todos los pesos al promedio de las actualizaciones de peso. Esto puede expresarse como:

$$\Delta w_{ij} = \frac{1}{N}\sum_{k=1}^{N} \Delta w_{ij}(k) \qquad (2.10)$$

Donde $\Delta w_{ij}(k)$ es la actualización de peso para el k-ésimo dato de entrenamiento y Nes el número total de datos de entrenamiento, este método es estable, pero es más lento que el método por SGD. En este método el número de ciclos de entrenamiento de la red es igual a una época. El error entregado para la actualización de pesos se muestra en la Fig. 2.8.

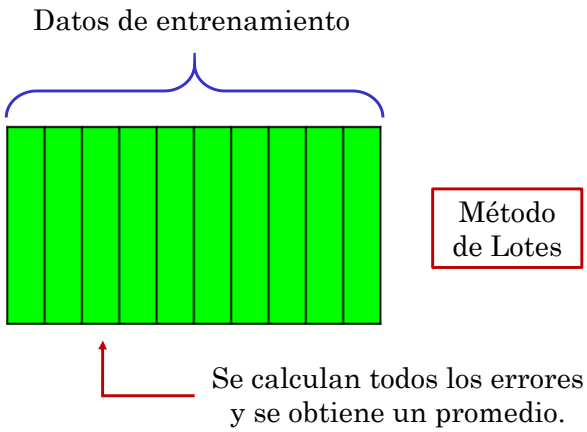

Fig. 2.8. El método por Lotes calcula la actualización de pesos para cada dato de entrenamiento, los promedia y modifica los pesos de la red con este valor promedio.

El método de Mini Lote es similar al método por Lotes, la diferencia es que crea subconjuntos del grupo de datos de entrenamiento, utiliza el método de Lotes con estos grupos y calcula la actualización de pesos por el método SGD, es decir calcula la actualización de pesos para este conjunto de datos, promedia estas actualizaciones y actualiza los pesos de la red y se repite este proceso con cada subconjunto de datos. Este método aprovecha las ventajas de los dos métodos anteriores, la velocidad del SGD y la estabilidad del método por Lotes. En este método se actualizan los pesos m veces (m es el número de subconjuntos de datos), entonces realiza m actualizaciones para el conjunto de datos de entrenamiento. La Fig. 2.9 muestra el agrupamiento de los datos de entrenamiento para la actualización de pesos.

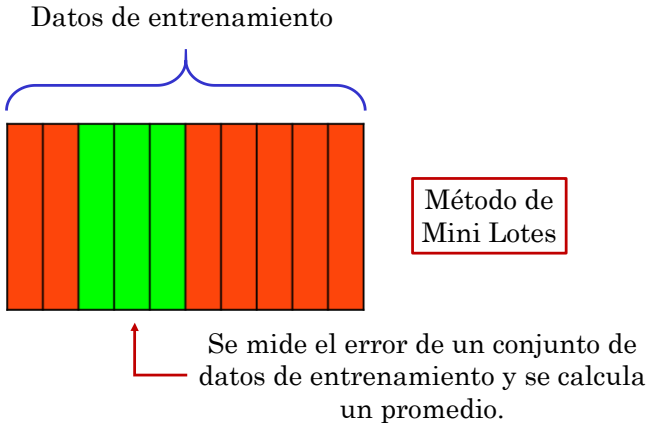

Fig. 2.9. El método de Mini Lote es una combinación del método GDE y el método de Lotes, este divide a los datos de entrenamiento en subgrupos y realiza el método de Lotes con estos grupos.

A continuación, se muestra un ejemplo sencillo de la implementación de los métodos descritos anteriormente.

2.2.4 Implementación de los métodos SGD, por Lotes y por Mini Lotes

Veamos ahora como pueden implementarse los métodos SGD, por Lotes y por Mini Lotes. Supongamos que tenemos el conjunto de datos que se muestra en la Tabla 2.1. Deseamos entrenar un Red Neuronal para obtener la salida correcta que se muestra en la misma tabla. Para implementar la técnica de SGD deberíamos seguir el siguiente pseudocódigo:

Para todos los datos de entrada se sigue el siguiente algoritmo:

i. *Calcular las salidas pesadas, Ecuación (2.2)*
ii. *Calcular la función de activación, Ecuación (2.3) con una sigmoide.*
iii. *Calcular el error, Ecuación (2.6b)*
iv. *Se determina el valor de delta, (2.7a)*
v. *Actualizar los pesos de la red (2.7)*

vi. *Repetir los pasos del i-v hasta alcanzar el error deseado.*

Tabla 2.1. Conjunto de datos de entrenamiento para la implementación de los métodos SGD, por Lotes y por Mini Lotes.

Datos de entrada	Datos de salida correcta
{0,0,1}	{0}
{0,1,1}	{0}
{1,0,1}	{1}
{1,1,1}	{1}

Por otra parte, para implementar el método por lotes debemos realizar lo siguiente:

Para todos los datos de entrenamiento se sigue el siguiente algoritmo:

i. *Calcular las salidas pesadas, ecuación (2.2)*
ii. *Calcular la función de activación, ecuación (2.3) con una sigmoide.*
iii. *Calcular el error, ecuación (2.6)*
iv. *Se determina el valor de delta, ecuación (2.7)*
v. *Se obtienen las actualizaciones de peso de la ecuación (2.7)*
vi. *Promediar los valores de las actualizaciones de peso como en la ecuación (2.10).*
vii. *Actualizar los pesos de la red ecuación (2.7)*
viii. *Repetir los pasos del i-vii hasta alcanzar el error deseado.*

Finalmente, para implementar el método de Mini Lote se deben seguir los siguientes pasos:

i. *Definir los conjuntos en que se dividirán a los datos de entrenamiento.*
ii. *Calcular las salidas pesadas, ecuación (2.2).*
iii. *Calcular la función de activación, ecuación (2.3) con una sigmoide.*
iv. *Calcular el error, ecuación (2.6).*
v. *Se determina el valor de delta, ecuación (2.7).*
vi. *Se obtienen las actualizaciones de peso de la ecuación (2.7).*
vii. *Se repiten los puntos del ii-vi m veces.*
viii. *Promediar los valores de las actualizaciones de peso para cada grupo de datos como en la ecuación (2.10).*
ix. *Actualizar los pesos de todos los nodos con este valor como en la ecuación (2.7).*
x. *Repetir los pasos anteriores para el siguiente grupo de datos.*
xi. *Repetir los pasos del i-x hasta alcanzar el error deseado.*

2.2.5 Resultados de la implementación de los métodos SGD y de Lotes

A continuación, mostraremos los resultados del entrenamiento de esta Red Neuronal compuesta por una sola capa oculta con los datos presentados en la Tabla 2.1.

Como se puede observar de la Fig. 2.17 a medida que se corrige el error entre la salida de la red y la salida correcta, este error disminuye. Después de 1000 épocas puede verse que el error es lo suficientemente pequeño. Como puede observarse, después de este entrenamiento los resultados que produce esta red son aceptables tal y como se muestran en la Tabla 2.2.

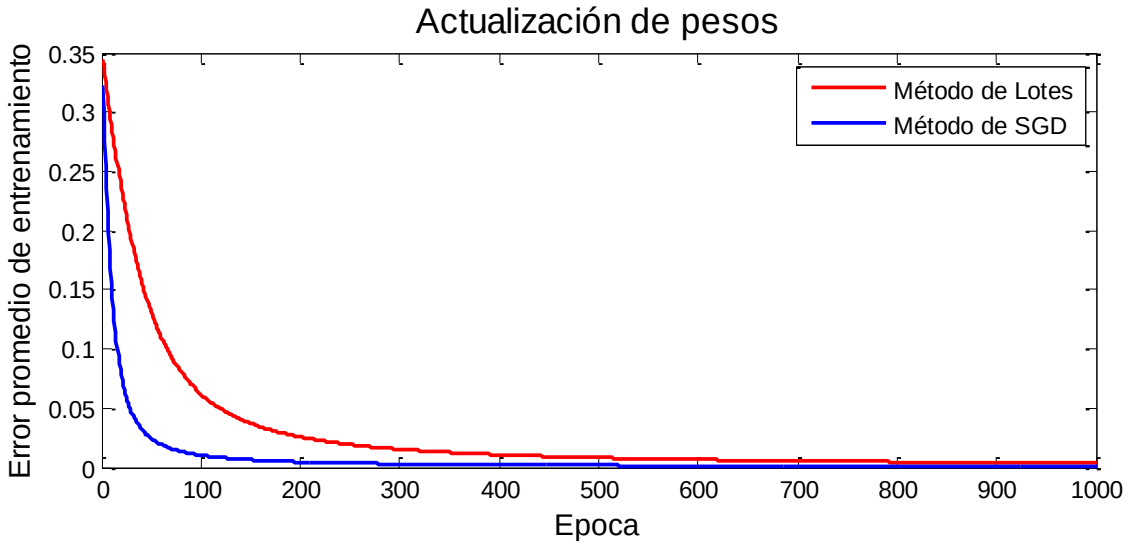

Fig. 2.10. Error promedio en el entrenamiento de una Red Neuronal Monocapa mediante el método SGD. En esta grafica se puede observar el comportamiento del error promedio a medida que se entrena la Red Neuronal. Como era de esperarse, el método SGD es más rápido que el método por lotes.

Tabla 2.2. Salidas de la red entrenada después de 1000 épocas. En la primera columna se muestran los datos de entrada de la red mientras que en la segunda y tercera columna se muestran la salida de la red y el valor correcto. El error absoluto entre ambas salidas es del orden 1%.

Datos de entrada	Salida de la red	Salida correcta
$\begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{1} & \mathbf{1}\end{bmatrix}$	$\begin{bmatrix}0.0102\\ 0.0083\\ 0.9932\\ 0.9917\end{bmatrix}$	$\begin{bmatrix}0\\ 0\\ 1\\ 1\end{bmatrix}$

2.2.6 Implementación del método SGD para el ejemplo XOR

En la sección anterior vimos la implementación del método SGD para un ejemplo sencillo, ahora vamos a utilizar este método para analizar el problema XOR, el nombre XOR es porque los datos de entrenamiento para este problema son similares a una tabla de verdad de una compuerta lógica XOR. En la Tabla 2.3 se muestran los datos de entrada y las respuestas correctas para la implementación de esta red. Si

utilizamos la red anterior y la entrenamos durante 1000 épocas, obtenemos los resultados mostrados en la Tabla 2.4.

Tabla 2.3. Conjunto de datos de entrenamiento para el ejemplo XOR, utilizando el método SGD

Datos de entrada	Salida correcta
{0,0,1}	{0}
{0,1,1}	{1}
{1,0,1}	{1}
{1,1,1}	{0}

Tabla 2.4. Salidas de la red entrenada después de 1000 épocas. En la primera columna se muestran los datos de entrada de la red mientras que en la segunda y tercera columna se muestran la salida de la red y el valor correcto. Es claro que hay una gran diferencia entre la salida correcta y la salida real de la red.

Datos de entrada	Salida de la red	Salida correcta
$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}$	$\begin{bmatrix} 0.5297 \\ 0.5000 \\ 0.4703 \\ 0.4409 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

¿Por qué la implementación del problema XOR arrojo resultados tan diferentes a la salida correcta? Para contestar esta pregunta grafiquemos los datos de entrenamiento del problema anterior.

Para esto, utilizaremos como coordenadas (x, y) los dos primeros dígitos de los datos de entrada y el valor en estos puntos serán los datos de salida correcta. Ahora, intentemos trazar una curva que separe los puntos en dos conjuntos, uno con valor 1 y el otro con 0. La Fig. 2.11 muestra el resultado de este análisis.

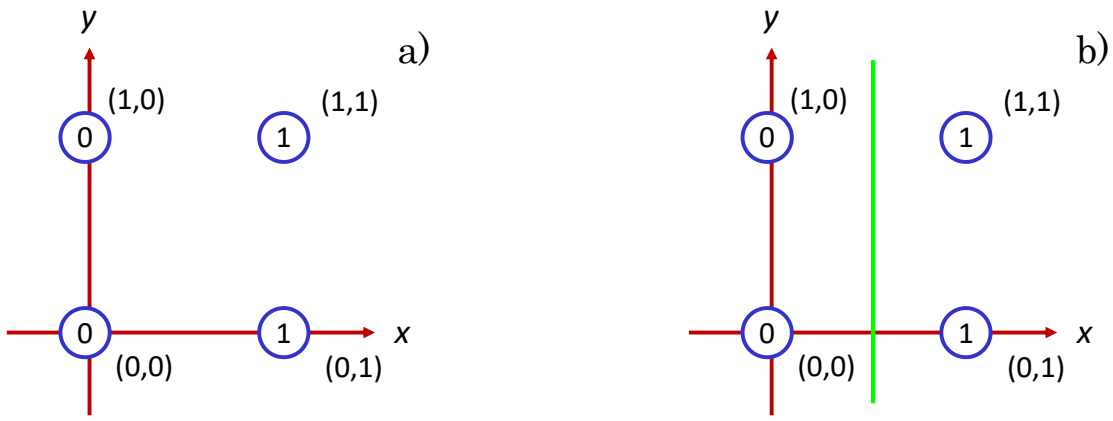

Fig. 2.11. Representación gráfica de los datos de entrenamiento del problema analizado usando el método SDG. En a) se "grafican" los datos de entrenamiento y en b) se separan los datos en dos grupos, uno con valor 1 y el otro con valor 0.

Da esta figura se puede observar que es posible separar los datos con una línea recta. Ahora analicemos los datos de entrenamiento para el problema XOR. La Fig. 2.12 muestra el resultado del análisis de estos datos.

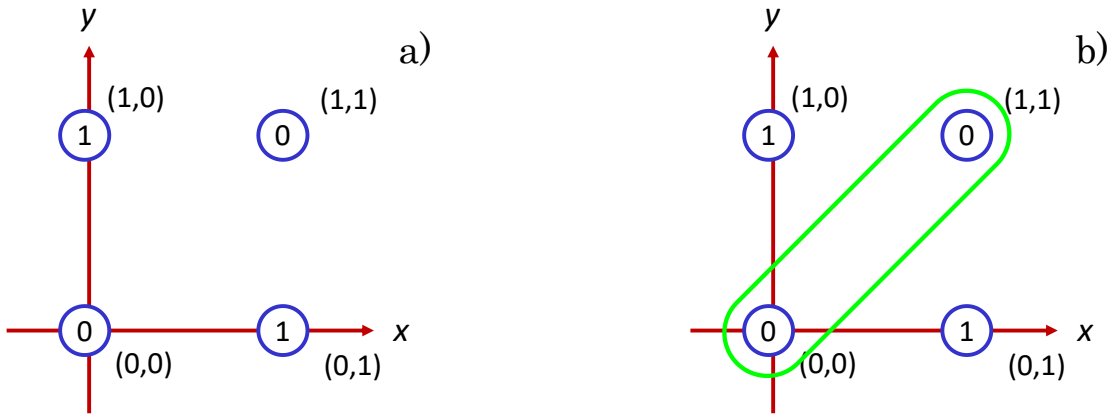

Fig. 2.12. Representación gráfica de los de los datos de entrenamiento del problema analizado usando el método SDG. En a) se "grafican" los datos de entrenamiento y en b) se separan los datos en dos grupos, uno con valor 1 y el otro con valor 0.

En la Fig. 2.12 se puede ver que para separar los datos se utiliza una curva complicada. Entonces se puede concluir que una Red Neuronal de una sola capa funciona para problemas relativamente sencillos, mientras que para problemas un poco más complejos esta red no funciona más. Es por esto que, se crean redes neuronales con más de una capa para resolver este tipo de problemas. A continuación, se describe el entrenamiento de una Red Neuronal Multicapa

2.3 ENTRENAMIENTO DE UNA RED NEURONAL MULTICAPA

El desarrollo de la Red Neuronal Multicapa tardo algunos años debido a que no se podía utilizar la Regla Delta porque no se conoce el error en las capas ocultas que es el elemento esencial para aplicar esta regla para el entrenamiento, esto provocó el desarrollo del algoritmo de propagación hacia atrás que se utiliza para conocer el error en las capas ocultas y así poder utilizar la Regla Delta Generalizada para entrenar la red. A continuación, se describe este algoritmo.

2.3.1 Algoritmo de propagación inversa

El algoritmo de propagación inversa determina el error en las capas ocultas para poder utilizar la Regla Delta Generalizada y poder ajustar los pesos para entrenar a la red. Supongamos que tenemos una Red Multicapa con una capa oculta con dos nodos, dos nodos de entrada y dos en la capa de salida tal y como se muestra en la Fig. 2.13.

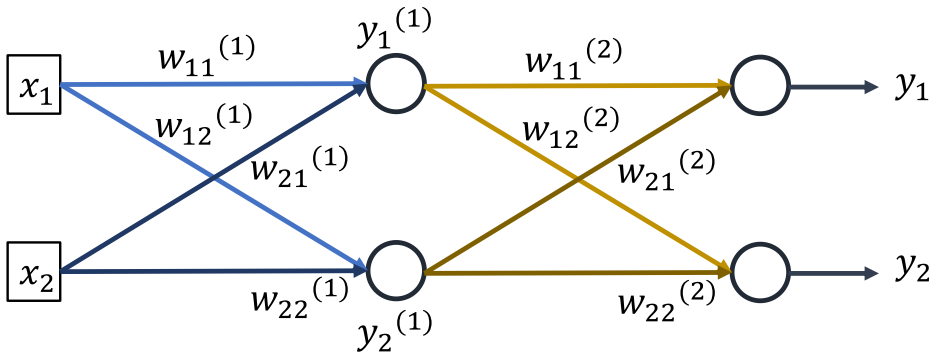

Fig. 2.13. Red Superficial, donde $w_{ij}^{(1)}$ son los pesos que hay entre la capa de entrada y la capa oculta, $y_k^{(1)}$ es la salida de la capa oculta, $w_{ij}^{(2)}$ son los pesos entre la capa oculta y la capa de salida y $y_{1,2}$ son las salidas de la red.

La salida de la capa oculta se puede obtener de la siguiente manera:

$$V^{(1)} = \begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = W^{(1)}X = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.11}$$

$$Y^{(1)} = \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} \varphi\left(v_1^{(1)}\right) \\ \varphi\left(v_2^{(1)}\right) \end{bmatrix} \tag{2.11}$$

Donde X esl vector de los nodos de entrada, $V^{(1)}$ es el vector de la suma ponderada de la primera parte, $Y^{(1)}$ el vector de salida de los nodos de la capa oculta y $W^{(1)}$ el vector de pesos entere la capa de entrada y la capa oculta.

De forma similar de obtiene la salida de la red de la siguiente forma.

$$V^{(2)} = \begin{bmatrix} v_1^{(2)} \\ v_2^{(2)} \end{bmatrix} = W^{(2)}Y^{(1)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} \tag{2.12}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \varphi\left(v_1^{(2)}\right) \\ \varphi\left(v_2^{(2)}\right) \end{bmatrix} \tag{2.12a}$$

$$D = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{2.12b}$$

Donde Y es el vector de salida de la red neuronal y D es el vector de salida con los datos correctos de la red.

Ahora podemos aplicar el algoritmo de propagación inversa, podemos calcular el error E entre D e Y, posteriormente se calcula el vector Δ.

$$E = D - Y = \begin{bmatrix} d_1 - y_1 \\ d_2 - y_2 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{2.13}$$

$$\Delta = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \varphi'(v_1)e_1 \\ \varphi'(v_2)e_2 \end{bmatrix} \tag{2.13a}$$

De aquí obtenemos la delta de la capa de salida y ahora podemos calcular la delta de los nodos de la capa oculta. En este algoritmo el error del nodo de define como la suma ponderada de los deltas propagados de derecha a izquierda.

$$e_1^{(1)} = w_{11}^{(2)}\delta_1 + w_{21}^{(2)}\delta_2 \tag{2.14}$$

$$\delta_1^{(1)} = \varphi'\left(v_1^{(1)}\right)e_1^{(1)} \tag{2.14a}$$

$$e_2^{(1)} = w_{12}^{(2)}\delta_1 + w_{22}^{(2)}\delta_2 \tag{2.14b}$$

$$\delta_2^{(1)} = \varphi'\left(v_2^{(1)}\right)e_2^{(1)} \tag{2.14c}$$

De forma matricial se puede expresar como:

$$E^{(1)} = \begin{bmatrix} e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \tag{2.15}$$

$$E^{(1)} = W^{(2)T}\Delta \tag{2.15a}$$

En la Ecuación (2.15) se puede ver que aparece la matriz W traspuesta, entonces se puede reescribir como se muestra en la Ecuación (2.15a).

Este proceso inicia en la capa de salida y se repite para todas las capas ocultas y termina en la primera capa oculta, se propaga de derecha a izquierda. Una vez calculados todas las deltas podemos ocupar la Ecuación (2.7).

2.3.2 Implementación de un ejemplo de propagación inversa

Vamos ahora a implementar un ejemplo sencillo del algoritmo de propagación inversa, resolveremos el problema que no pudo ser resuelto mediante el uso de una red de una sola capa que fue mostrado en la sección 2.2.6. Por conveniencia se repite nuevamente los datos de entrada y salida correcta de la función XOR en la Tabla 2.5.

Tabla 2.5. Datos de entrada y la salida correcta para la implementación de la función lógica XOR.

Datos de entrada	Salida correcta
{0,0,1}	{0}
{0,1,1}	{1}
{1,0,1}	{1}

{1,1,1}	{0}

Vamos a considerar que nuestra red neuronal consiste de dos nodos de entrada, y un solo nodo de salida. Por otra parte, vamos a utilizar una capa oculta con cuatro nodos de salida mientras que la última capa, como ya se mencionó, solo tiene un nodo de salida. El esquema de esta Red Neuronal se muestra en la Fig. 2.14.

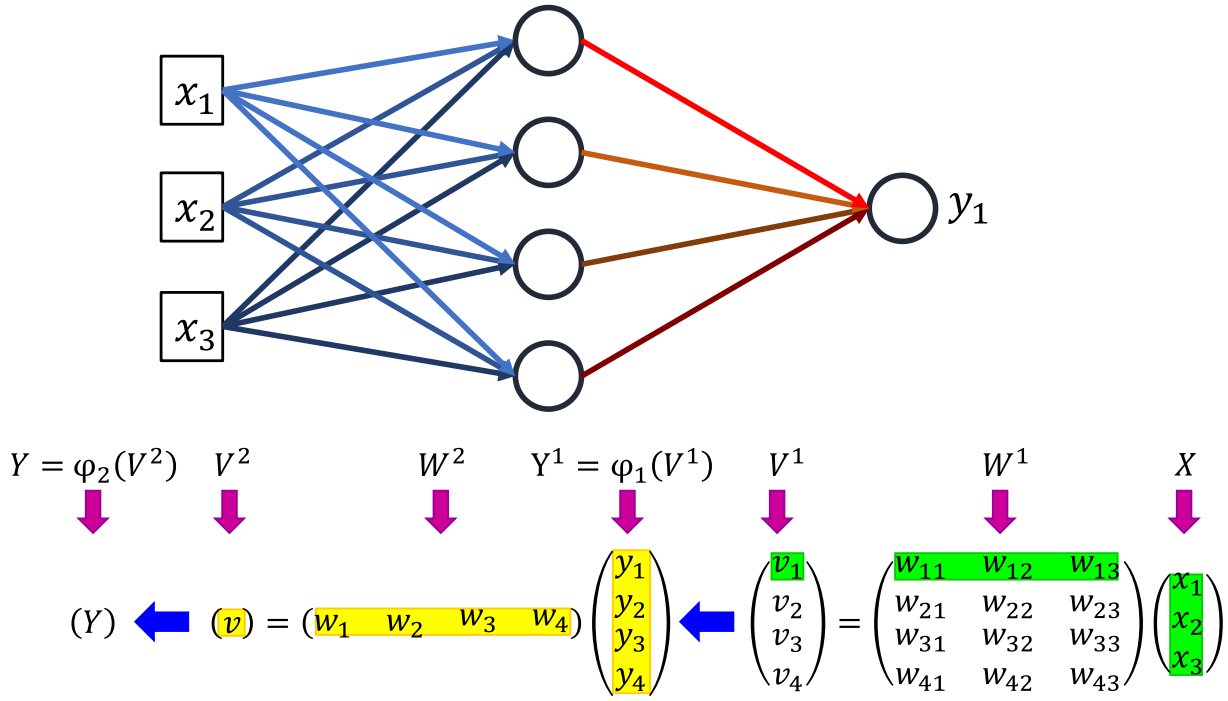

Fig. 2.14. Red de trabajo que contiene una apa oculta y una capa de salida. Se consideran 3 nodos de entrada y un solo nodo de salida.

Vamos a utilizar la función sigmoide como función de activación tanto para los nodos ocultos como para el nodo de salida, emplearemos además la SGD junto con el algoritmo de propagación inversa.

Para hacer esta implementación vamos a utilizar el siguiente algoritmo:

i. *Inicializar los pesos con valores adecuados*
ii. *Tomar los datos de entrada a partir de los datos de entrenamiento y obtener la salid de la red de trabajo.*
iii. *Calcular el error entre la salida y la salida correcta y obtener δ de la síguete forma: $e = d - y$, $\delta = \varphi'(v)e$*
iv. *Propagar la delta del nodo en propagación inversa, hacia las capas ocultas y calcular las deltas de los nodos inmediatamente a la izquierda: $e^{(k)} = W^T\delta$, $\delta^{(k)} = \varphi'\left(v^{(k)}\right)e^{(k)}$*
v. *Repetir el paso iv hasta alcanzar la capa oculta que esta inmediatamente a la derecha de la capa de entrada.*
vi. *Ajustar los pesos de acuerdo con la siguiente regla de aprendizaje: $\Delta w_{ij} = \alpha\delta_i x_j$, $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$.*
vii. *Repetir los pasos ii-vi para cada dato de entrenamiento*

viii. *Repetir los pasos ii-vii hasta que la red tenga un entrenamiento adecuado.*

Tabla 2.6. Resultados del entrenamiento de un Red Multicapa para la función XOR.

Datos de entrada	Salida de la red	Salida correcta
$\begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{1} & \mathbf{1}\end{bmatrix}$	$\begin{bmatrix}0.0060\\ 0.9888\\ 0.9891\\ 0.0134\end{bmatrix}$	$\begin{bmatrix}0\\ 1\\ 1\\ 0\end{bmatrix}$

Como puede verse ahora es posible obtener la salida correcta, la red entrenada tiene un error del orden de 0.01% en la estimación de la salida correcta. En la Fig. 2.15 se muestra cómo cambia el error de entrenamiento promedio en función del número de épocas.

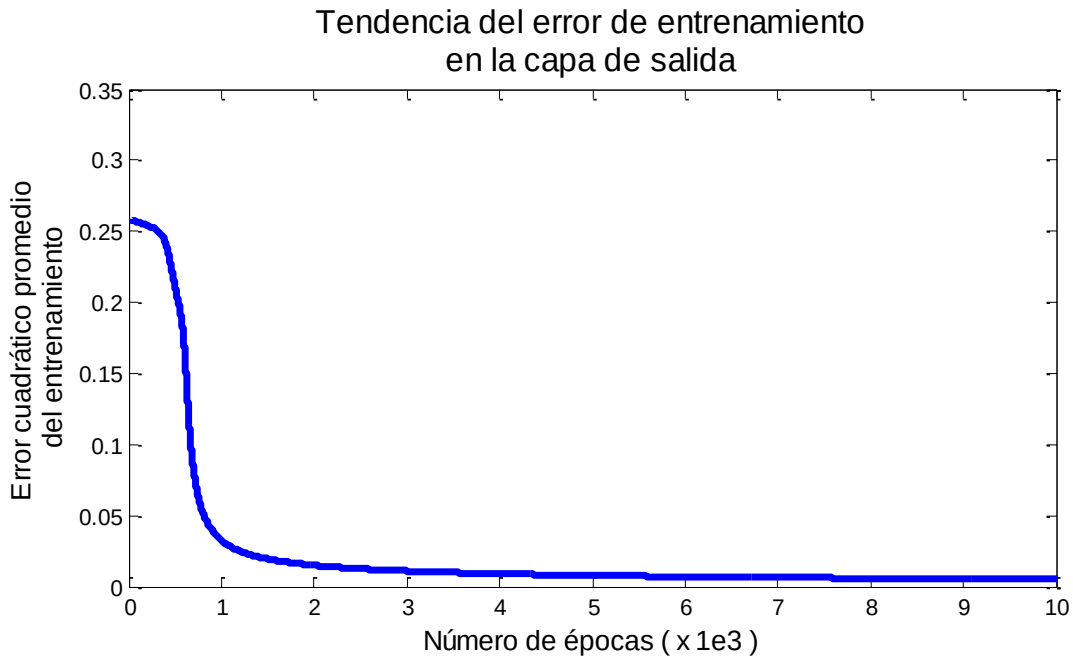

Fig. 2.15. Error de entrenamiento promedio para la implementación del problema XOR.

Como puede observarse, el error se encuentra por debajo del 5% y se logra aproximadamente después de 2000 épocas de entrenamiento. Algo interesante de esta grafica es que el error cae dramáticamente en las primeras épocas de entrenamiento mientras que al final el cambio en este error es mínimo. En la Fig. 2.16 se muestran los errores promedio en los coeficientes de la matriz de pesos en la capa oculta, como puede observarse, la evolución del error es similar al de la salida de la red. Las diferencias entre los signos de estos errores se deben básicamente a que en el inicio del entrenamiento los valores de los coeficientes estaban por encimo o por debajo del valor correcto.

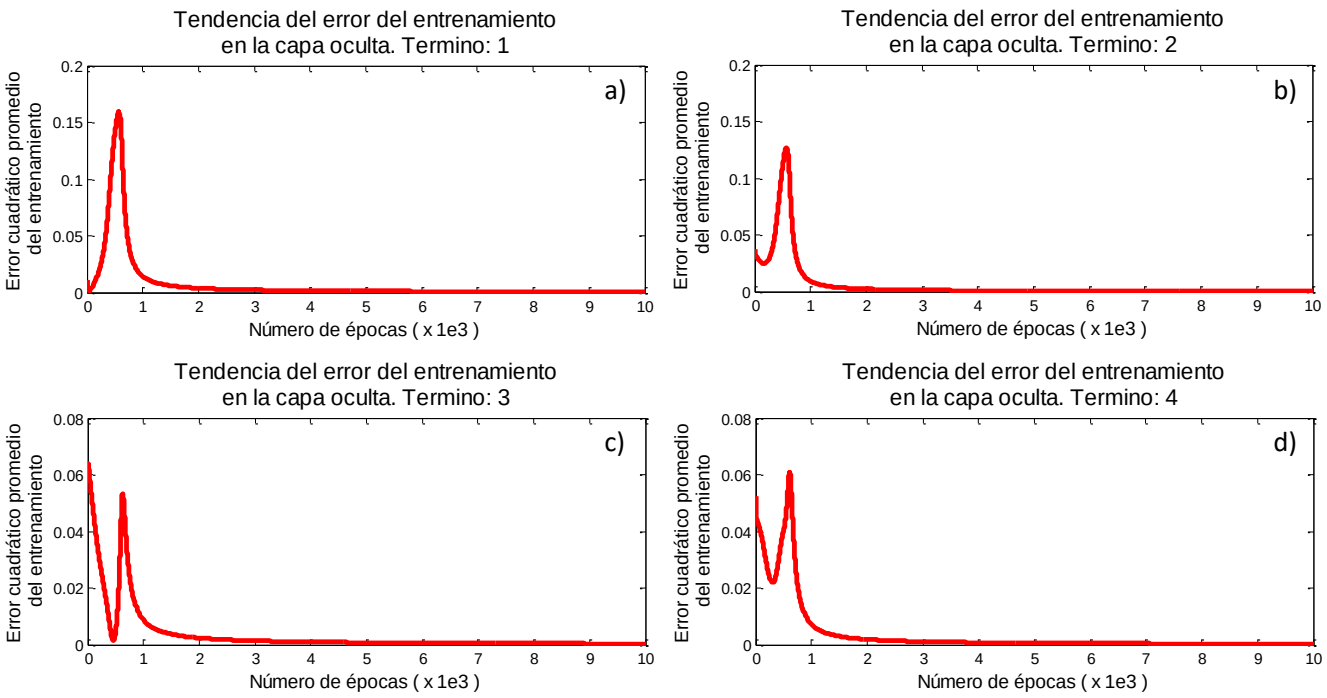

Fig. 2.16. Error promedio en el entrenamiento de la Red Multicapa en los coeficientes de la matriz de pesos para la implementación del problema XOR. Comportamiento del error en los coeficientes: a) w_{11}, b) w_{12}, c) w_{21} y d) w_{22}.

En conclusión, es claro que esta Red Multicapa puede resolver el problema de la XOR el cual no pudo ser resuelto por una Red Monocapa. Hasta ahora se ha ajustado el peso de la forma más simple utilizando la Ecuación (2.6) y la Ecuación (2.7), sin embargo, existen diferentes formas de realizar la actualización de estos pesos. Vamos a mostrar ahora otro algoritmo que optimiza el desempeño del SGD. Este algoritmo es conocido como el algoritmo del *momento,* el cual básicamente anexa un término, *m,* que se suma a la Regla Delta para ajustar el peso en cierta dirección.

$$\Delta w = \alpha \delta x \tag{2.16}$$

$$m = \Delta w + \beta m \tag{2.16a}$$

$$w = w + m \tag{2.16b}$$

$$m^{-} = m \tag{2.16c}$$

Donde m es el término de "impulso", este contiene la información de la actualización de pesos de la época anterior y β es una constante entre 0 y 1 que determina el grado de influencia de m en la actualización (es decir, como las mediciones anteriores afectan al valor actual de los pesos).

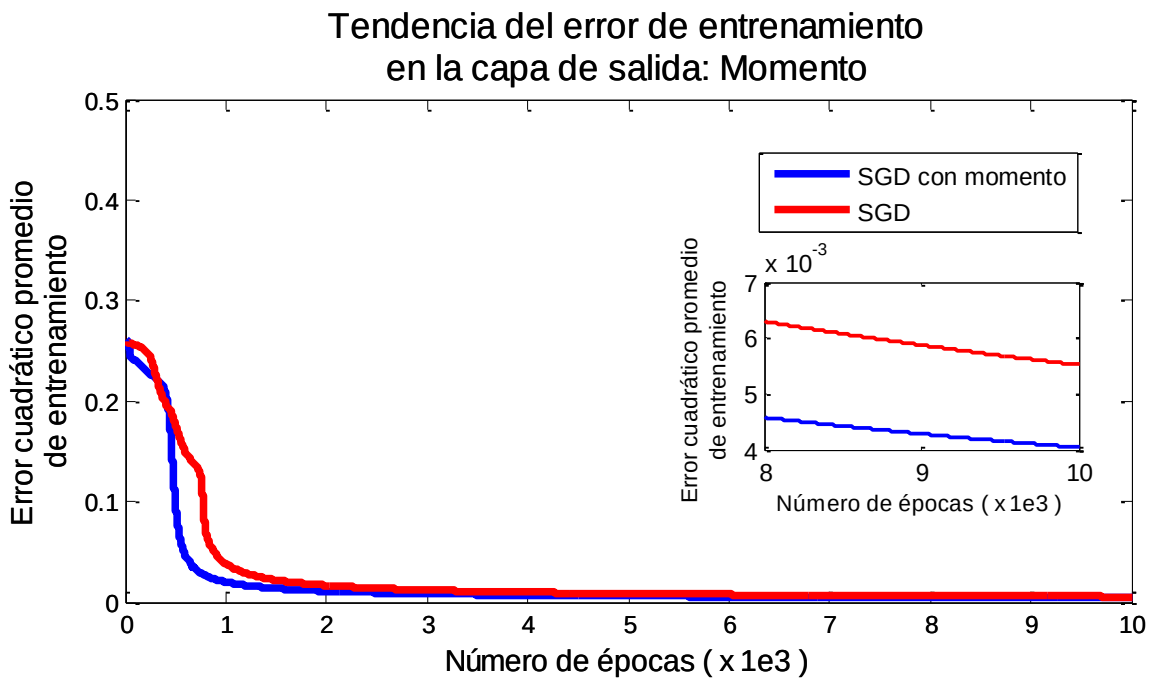

Fig. 2.17. Comportamiento del error promedio de la salida de la Red Neuronal mediante el método SGD y el método SGD con momento. En estas graficas se puede observar el comportamiento del error promedio a medida que se entrena la Red Neuronal.

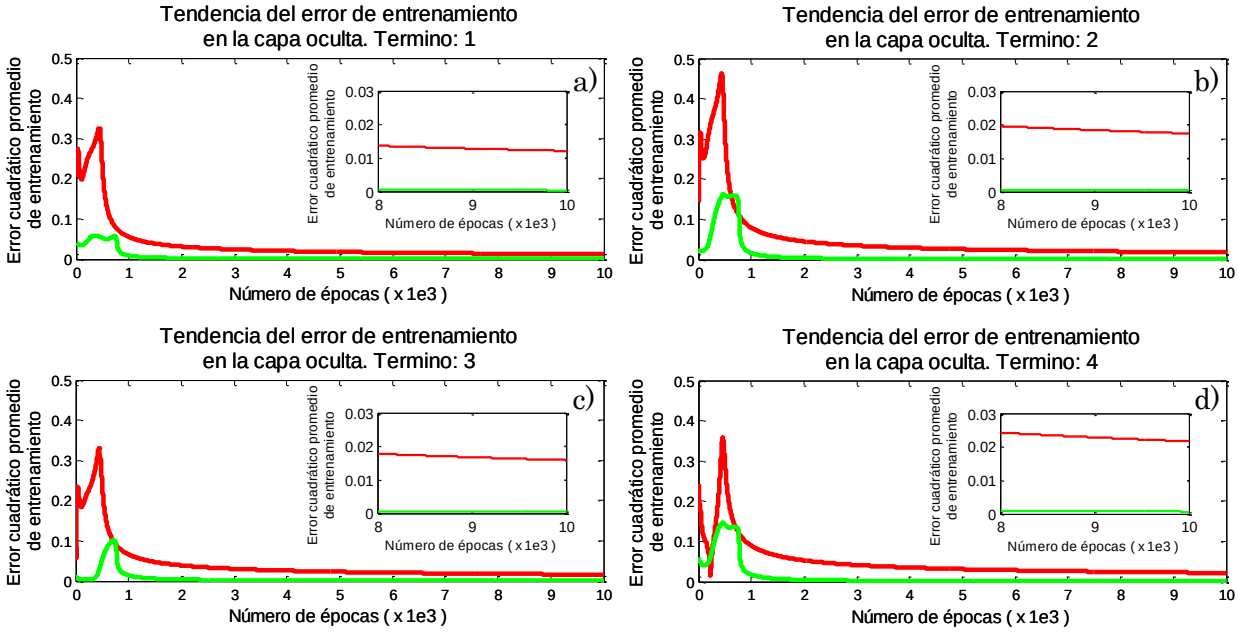

Fig. 2.18. Comportamiento del error promedio en el entrenamiento de la Red Neuronal mediante el método SGD en color verde y el método SGD con momento en color rojo. En estas graficas se puede observar el comportamiento del error promedio de las salidas de los nodos de la capa ocultas a medida que se entrena la Red Neuronal. a) Es el error del primer nodo de la capa oculta, b) para la segunda, c) para la tercera y d) para el ultimo nodo.

2.3.3 Función de costo y regla de aprendizaje

La función de costo es un concepto matemático asociado con la teoría de optimización, y está relacionada con el aprendizaje supervisado de la Red Neuronal. La función de costo es la medida del error de la red neuronal. Cuanto mayor es el error de la Red Neuronal, mayor es el valor de la función de costo. Hay dos representaciones comunes de la función de costo para el Aprendizaje Supervisado de la Red Neuronal. Estas funciones se muestran en las Ecuaciones (2.17).

$$j = \sum_{i=1}^{M} \frac{1}{2}(d_i - y_i)^2 \qquad (2.17)$$

$$j = \sum_{i=1}^{M} \{-d_i \ln(y_i) - (1 - d_i)\ln(1 - y_i)\} \qquad (2.17a)$$

La función de costo de la Ecuación (2.17) es la suma de los errores al cuadrado, si el error es pequeño, este será más pequeño, si d_i e y_i son iguales el error será cero, esta es proporcional al error.

La fórmula que está entre las llaves de la Ecuación (2.17a) se llama función de entropía cruzada. De acuerdo con la definición del logaritmo esta función está definida en el intervalo que esta entre 0 y 1, La Fig. 2.19 muestra una familia de graficas de la función de entropía cruzada para diferentes valores del parámetro d que varían entre 0 y 1.

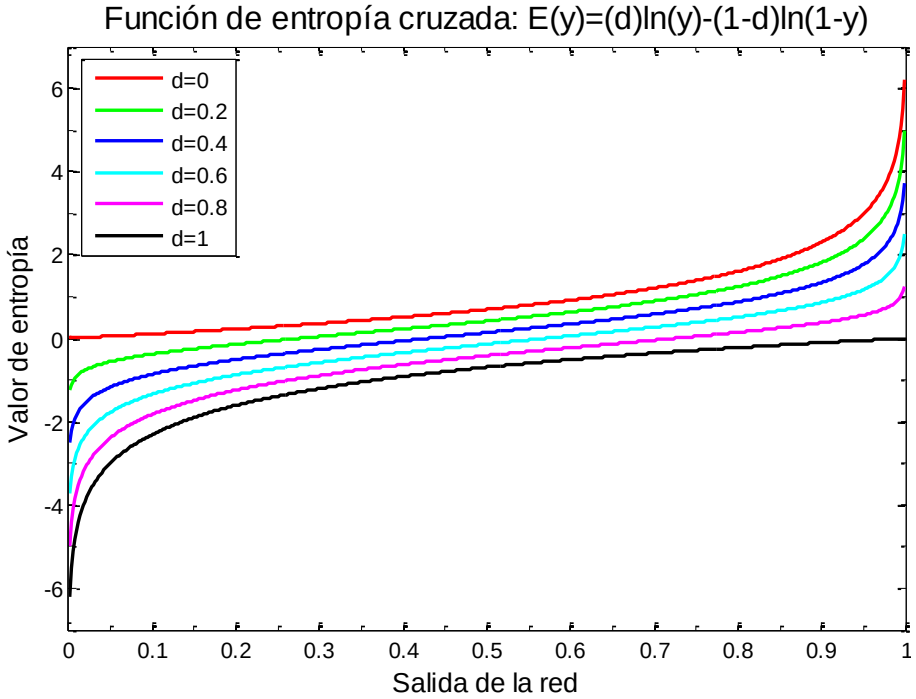

Fig. 2.19. La función de entropía cruzada evalúa el valor de la salida de la red. En la gráfica se muestran curvas para distintos valores del parámetro d.

La función de entropía cruzada a menudo se asocia con la función Sigmoide de la Ecuación

(2.8) y con la ecuación SoftMax.

La función de entropía cruzada es más sensible al error que la función cuadrática y por lo mismo se distingue por producir un mejor rendimiento.

La Fig. 2.20 muestra las gráficas obtenidas al implementar la función de entropía cruzada en el ejemplo del problema XOR.

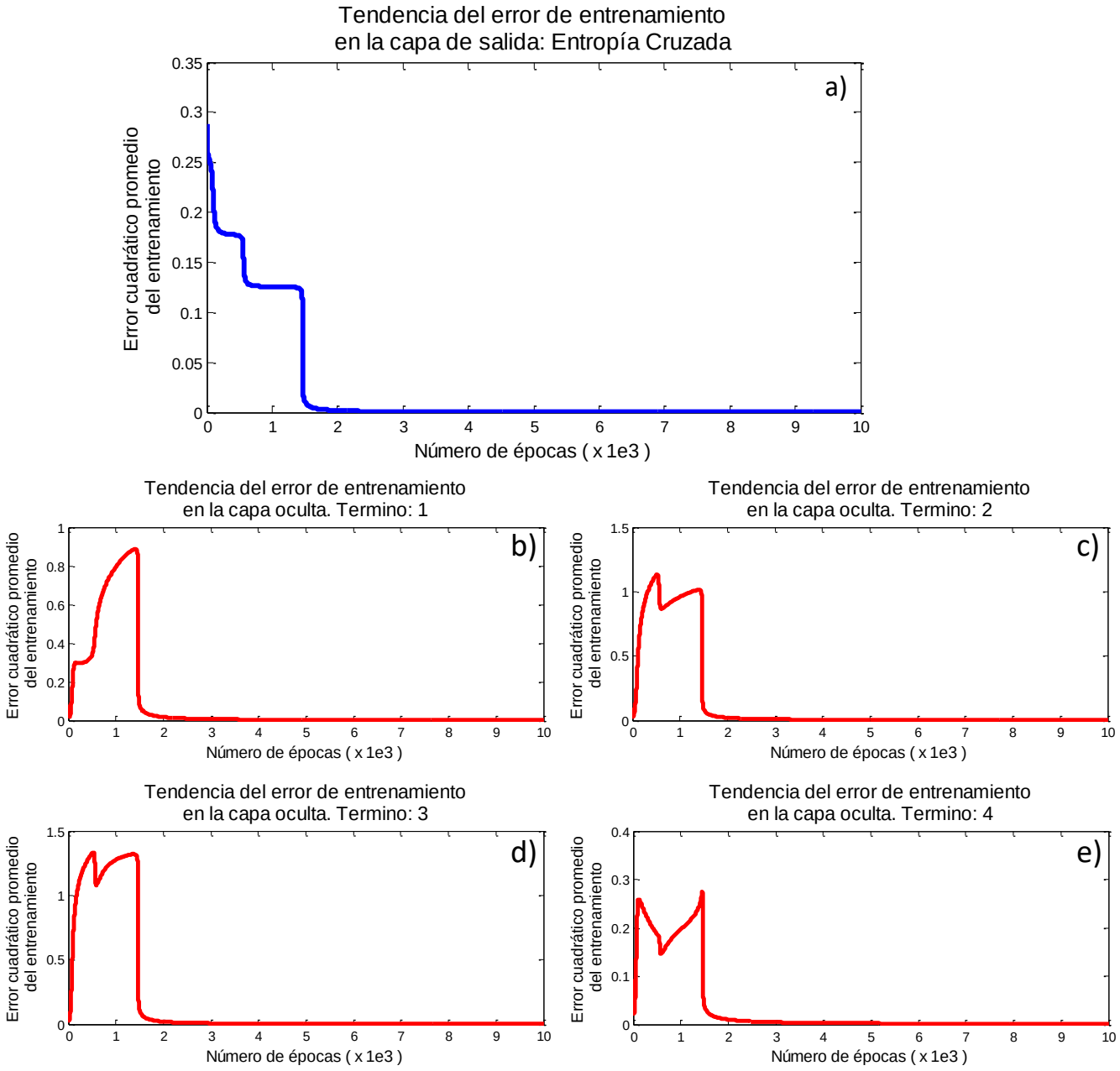

Fig. 2.20. Comportamiento del error promedio en el entrenamiento de la Red utilizando el método de entropía cruzada, utilizado para resolver el problema XOR, donde a) es la magnitud del error obtenido en la capa de salida, b) Es el error del primer nodo de la capa oculta, c) para la segunda, d) para la tercera y e) para el ultimo nodo.

La Fig. 2.21 muestra la comparación de los resultados obtenidos con el método SGD, SGD con momento y entropía cruzada

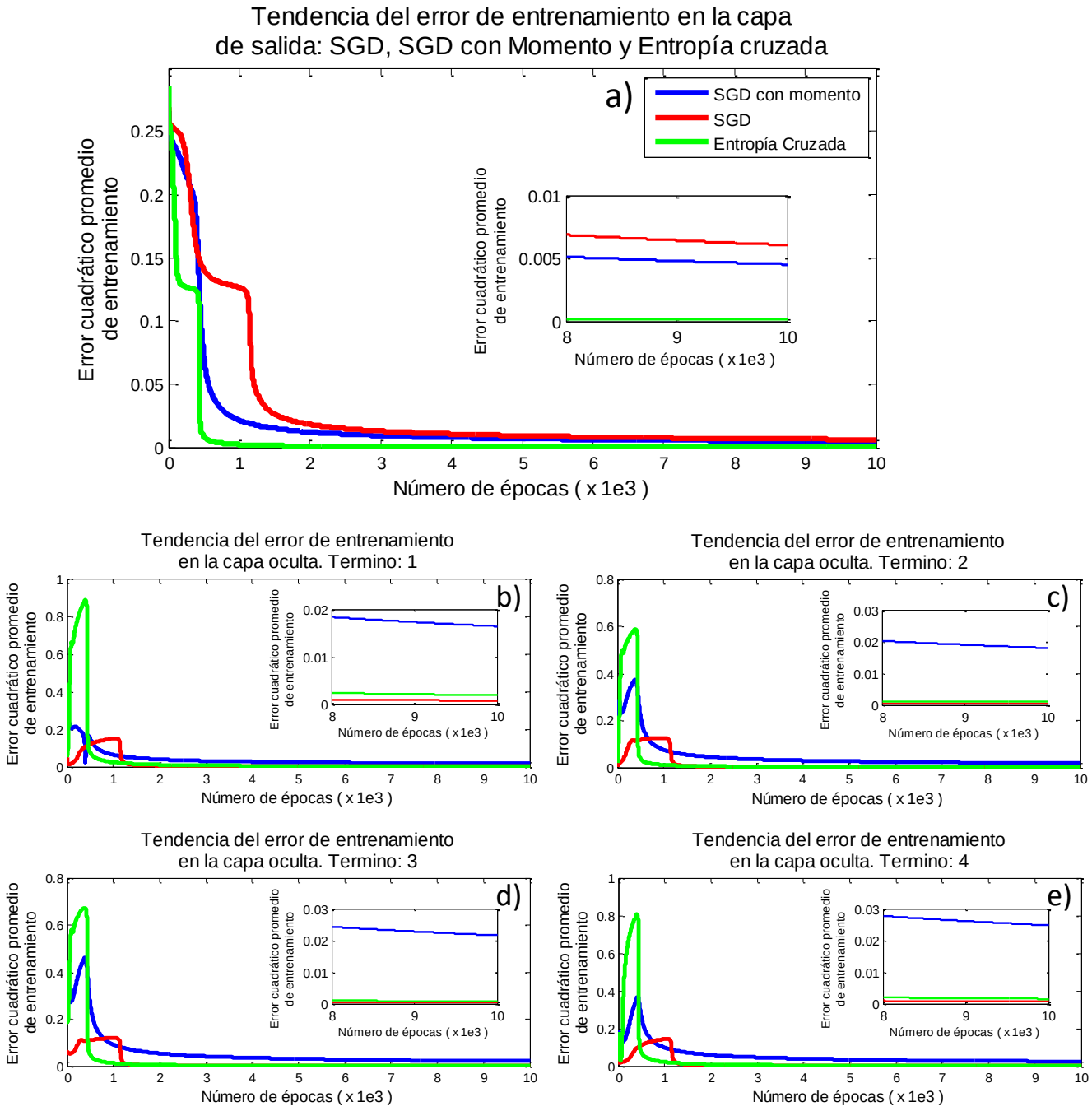

Fig. 2.21. Comportamiento del error promedio en el entrenamiento de la Red. Aquí se muestra una comparación de los resultados obtenidos utilizando los métodos SGD, SGD con momento y entropía cruzada, utilizados para resolver el problema XOR, donde a) es la magnitud del error obtenido en la capa de salida, b) Es el error del primer nodo de la capa oculta, c) para el segundo, d) para el tercero y e) para el último nodo.

Capítulo 3. Red Neuronal y clasificación

Este capítulo presenta un ejemplo de la clasificación, que es la aplicación más común de una Red Neuronal. La clasificación multiclase separa a los datos de entrada en más de dos grupos.

Como vimos anteriormente la clasificación es una de las principales aplicaciones del Aprendizaje Supervisado, el número de nodos de salida varía según el número de grupos que se desean diferenciar.

3.1 Clasificación multiclase

La clasificación multiclase separa los datos de entrada en más de dos grupos, lo más común para una Red Multiclase es igualar el número de nodos de salida al número de clases. Supongamos que deseamos determinar a partir de una matriz de 5x5 pixeles el digito decimal que esta dibujado, ver Fig. 3.1.

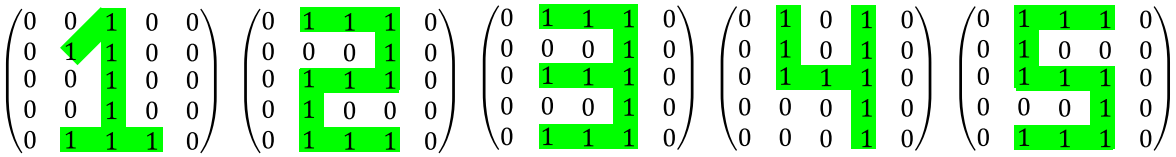

Fig. 3.1. Matrices binarias de 5x5 pixeles que representan los dígitos del 1-5.

Estas matrices de 5x5 pueden ser redimensionadas para representarse como un vector columna de 25x1 elementos. Por lo tanto, nuestra red multiclase contara con 25 nodos de entrada mientras que debemos tener cinco nodos de salida, tal y como se muestra en la Fig. 3.2. Vamos a incluir una capa oculta de 50 nodos y por lo tanto las matrices de pesos de la capa oculta y de salida tienen dimensiones de 50x25 y 5x50 respectivamente, tal y como se muestra en la Fig. 3.3.

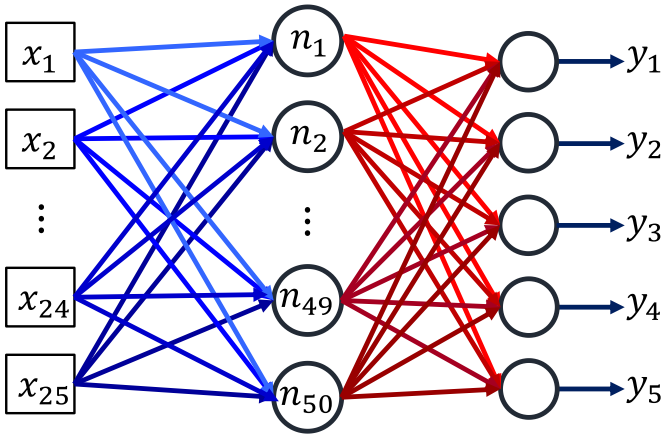

Fig. 3.2. Red Multiclase, esta tiene 25 nodos de entrad que corresponden a las entradas de las matrices utilizadas para el entrenamiento de los datos, 50 nodos en la capa oculta y cinco nodos de salida.

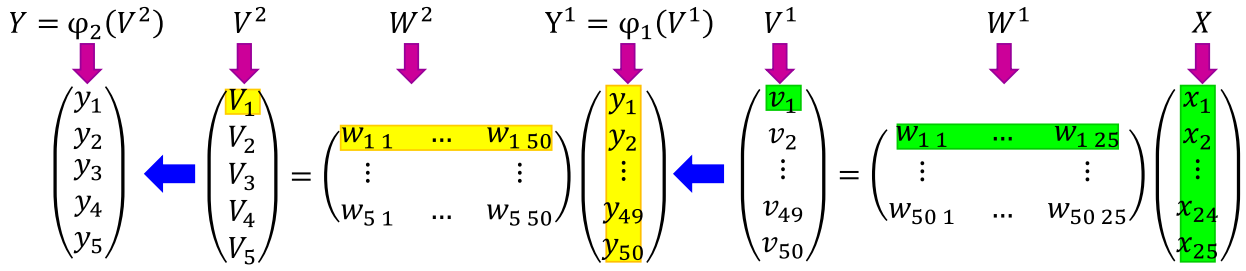

Fig. 3.3. Dimensiones de las matrices que conforman la Red Neuronal Multiclase de la Fig. 3.2.

Las clases de esta red pueden ser visualizadas como se muestra en la Fig. 3.4.

$$Clase\ 1 \rightarrow [1,0,0,0,0]^T$$

$$Clase\ 2 \rightarrow [0,1,0,0,0]^T$$

$$Clase\ 3 \rightarrow [0,0,1,0,0]^T$$

$$Clase\ 4 \rightarrow [0,0,0,1,0]^T$$

$$Clase\ 5 \rightarrow [0,0,0,0,1]^T$$

Fig. 3.4. Vector de salida de la red Multiclase para las distintas clasificaciones. El operador [ ]^T denota la traspuesta del vector.

Al ingresar datos a la entrada de la red, esta dará como resultado un 1 para alguno de los nodos de salida y cero para los demás, como se muestra en la Fig. 3.5.

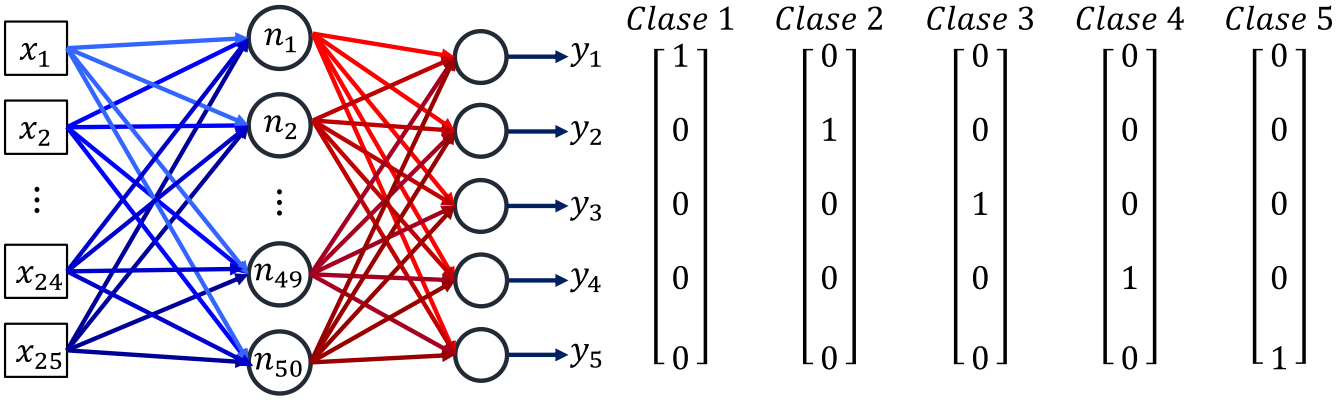

Fig. 3.5. En la clasificación para una Red Multiclase con N grupos, los datos se escriben como un vector de N nodos repletos de ceros y un 1 en la posición a la que pertenece el dato.

Esta técnica se llama "uno de *N*", donde *N* es el número de clases, y es por lo que, se hace coincidir el número de salidas con el de clases. Hasta ahora se ha utilizado la función Sigmoide, pero ésta solo contiene la información de la suma ponderada de un nodo, ahora es preferible utilizar la función SoftMax la cual contiene la información de todos los nodos de la capa. La función SoftMax se define como sigue:

$$y_i = \varphi(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + +e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}} \qquad (3.1)$$

Donde v_i es la suma ponderada para el i-ésimo nodo de salida y M es el número total de nodos en la capa.

Esta ecuación entrega los resultados normalizados a uno, es decir que la red entrega probabilidades de que el dato de entada pertenezca a uno de los grupos. La regla de aprendizaje usada para una red multiclase es la de entropía cruzada. A continuación, se muestran los resultados del entrenamiento de esta red.

3.1.1 Resultado del entrenamiento de la red multiclase

Como puede verse en la Fig. 3.6 los errores en las salidas caen dramáticamente a medida que se entrena a la red. Por esta razón, el error ha tenido que graficarse en escala logarítmica. Los valores finales de error se encuentran alrededor de 10^{-5}.

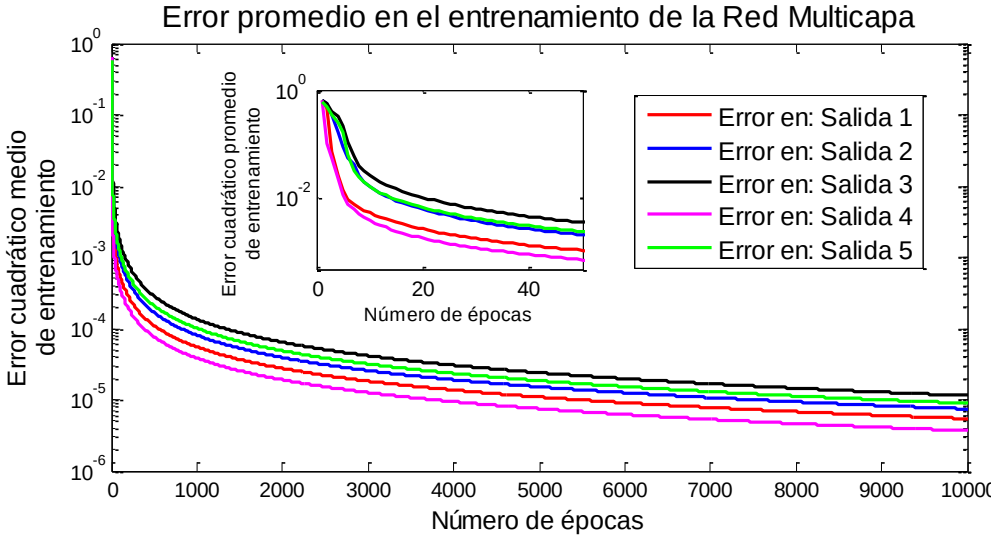

Fig. 3.6. Error de las cinco salidas en el entrenamiento de la red de clasificación multiclase.

Par evaluar el desempeño de esta red con datos "reales" se creó un programa con una interfaz de usuario grafica (GUI por sus siglas en inglés). En ella se pueden introducir con el ratón de la computadora, distintas matrices para que sean "reconocidas" por nuestra red neuronal.

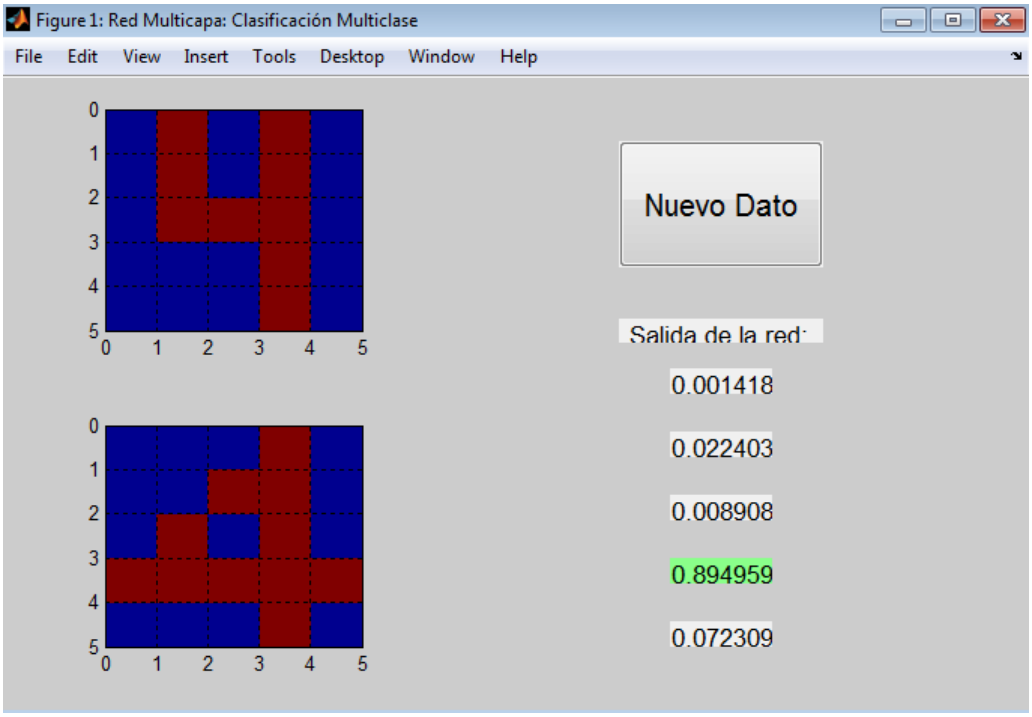

Fig. 3.7. GUI utilizada para el reconocimiento de dígitos utilizando una red multiclase. Para este caso se introdujo el dato de entrada y el programa entrego una probabilidad del 89.5% de que sea un 4, por lo que se toma como salida al digito 4.

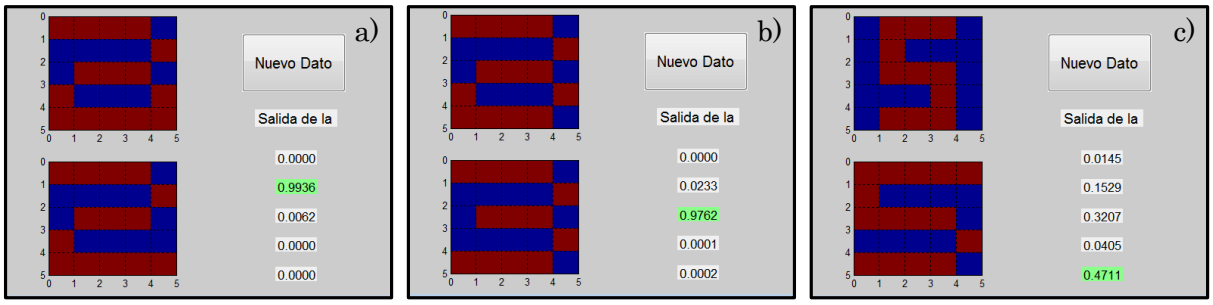

Fig. 3.8. Resultados de la red entrenada para tres datos de entrada distintos. a) En este caso la red entrenada reconoce el digito en la matriz como un 2. b) En el segundo caso el dato de entrada es reconocido como el digito 3 con una certeza del 97% y finalmente en c) el dato de entrada es reconocido como el digito 5 con una certeza del 47%, sin embargo, esta certeza es baja por lo que uno podría incluso tomar el segundo mejor valor de certeza que en este caso es 32% el cual corresponde al digito 3.

En la Fig. 3.8 se muestran tres datos de entrada diferentes con sendas salidas. Como puede observarse, en los primeros dos casos la red neuronal tiene un nivel de certeza superior al 90% por lo cual puede considerarse que el reconocimiento es adecuado. Sin embargo, en el último caso la certidumbre que muestra la red es menor al 50% por lo que aun cuando la GUI desarrollada da como resultado el digito 5 bien podría haberse tomado el segundo valor de salida más alto el cual corresponde a un nivel de certeza de 32% el cual corresponde al digito 3. Estos niveles tan bajos en la determinación de a qué clase pertenece este dato de entrada se deben a lo simple de la

red y a la falta de un mayor número de datos de entrenamiento. Mas adelante se mostrará un ejemplo con una red que ofrecerá resultados más satisfactorios.

Antes de este será necesario comprender los conceptos detrás del aprendizaje profundo, tema que se verá a continuación.

CAPÍTULO 4. APRENDIZAJE PROFUNDO

Con el paso de los años surgieron complicaciones al querer resolver problemas más complejos en donde era necesario incrementar el número de capas ocultas de una Red Neuronal. Al hacer esto, la red tiene un rendimiento deficiente, para solucionarlo se desarrolló una técnica llamada Aprendizaje Profundo.

La razón por la que la Red Neuronal con capas más profundas arrojó resultados más pobres fue que la red no estaba debidamente capacitada. El algoritmo de propagación hacia atrás experimenta las siguientes tres dificultades principales en el proceso de entrenamiento de la Red Neuronal Profunda:

- Gradiente de desaparición
- Sobreajuste
- Carga computacional

Este capítulo presenta estos problemas y muestra también la solución a estos.

4.1 GRADIENTE DE DESAPARICIÓN

El gradiente de desaparición (vanishing gradient en inglés) ocurre cuando al aplicar el algoritmo de propagación inversa, el error no llega a las capas del inicio de la red, cuando esto pasa, los pesos de estas capas no se ajustan adecuadamente y la red no puede ser entrenada. La solución para este problema es el uso de la función de Unidad de Rectificación Lineal (ReLU, por sus siglas en inglés) como una función de activación, ya que transmite mejor el error que la función Sigmoide, la función ReLU se define como se muestra en la ecuación (4.1) y su perfil se presenta en la Fig. 4.1

$$\varphi(x) = \begin{cases} 0 \; si \; x \leq 0 \\ x \; si \; x > 0 \end{cases} \tag{4.1}$$

$$\varphi(x) = max(0, x) \tag{4.1a}$$

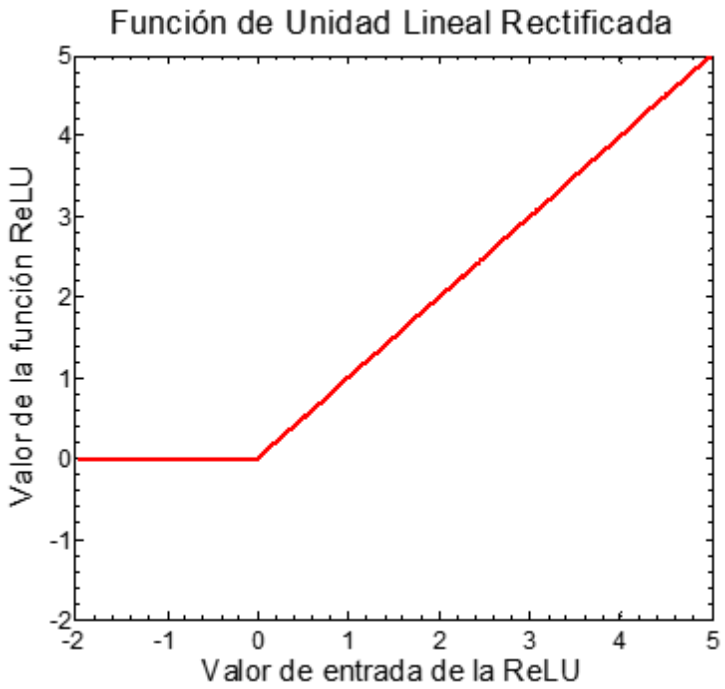

Fig. 4.1. La función de Unidad de Rectificación Lineal entrega como salida el valor máximo entre la entrada y cero, es decir si los datos son negativos, la salida es cero y si son positivos, la salida es el mismo dato de entrada.

La función ReLU considera solo las señales positivas a diferencia de la función sigmoide que toma los valores positivos y negativos. Esto soluciona el problema del gradiente de desaparición. Otro elemento que necesitamos para el algoritmo de propagación inversa es la derivada de la función ReLU.

$$\varphi'(x) = \begin{cases} 0 \; si \; x \leq 0 \\ 1 \; si \; x > 0 \end{cases} \tag{4.2}$$

4.2 SOBREAJUSTE

Como vimos anteriormente entre más complicada sea la red ésta más sensible al sobreajuste, al agregar más capas este problema se agrava. La solución a este problema es la "desactivación" (dropout en inglés) que inhibe algunos de los nodos de las capas al azar. Mientras que las salidas se establecen como ceros. Este proceso se ilustra en la Fig. 4.2.

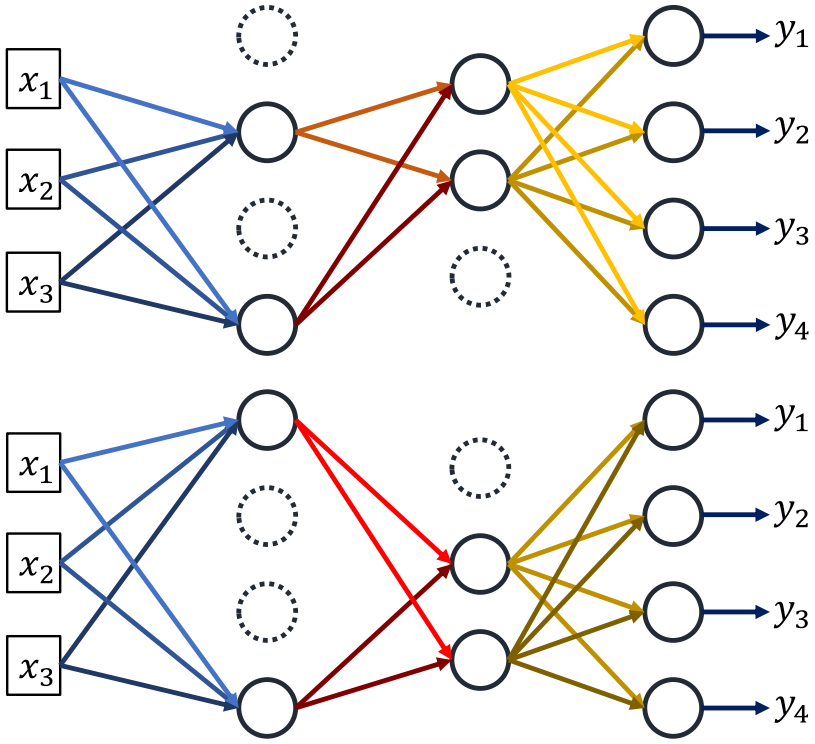

Fig. 4.2. El proceso de desactivación inhibe un grupo de nodos e introduce los datos para entrenar a la red como se muestra en la figura superior, después modifica aleatoriamente el grupo de nodos desactivados como se ve en la figura inferior y entrena la red, este proceso se hace en repetidas ocasiones para reducir el sobreajuste.

Los porcentajes usuales en la desactivación se encuentran aproximadamente entre 50% y 25% para capas ocultas y las capas de salida, respectivamente.

4.3 CARGA COMPUTACIONAL

Al aumentar el número de capas ocultas aumenta el número de pesos que hay que actualizar y se necesitan más datos de entrenamiento por lo que hay que realizar más cálculos lo que llevaran más tiempo para entrenar a la red. Este problema se soluciona con la introducción de hardware de alto rendimiento y algoritmos como la normalización por lotes.

A continuación, se realiza el entrenamiento de una red neuronal con el mismo ejemplo utilizado para la clasificación de dígitos.

4.4 EJEMPLO DE UNA RED DE CLASIFICACIÓN DE DÍGITOS CON APRENDIZAJE PROFUNDO

En este ejemplo vamos a considerar una red neuronal profunda con tres capas ocultas, tal y como se muestra en la Fig. 4.3. En la Fig. 4.4 se muestran las dimensiones de las matrices utilizadas en este ejemplo.

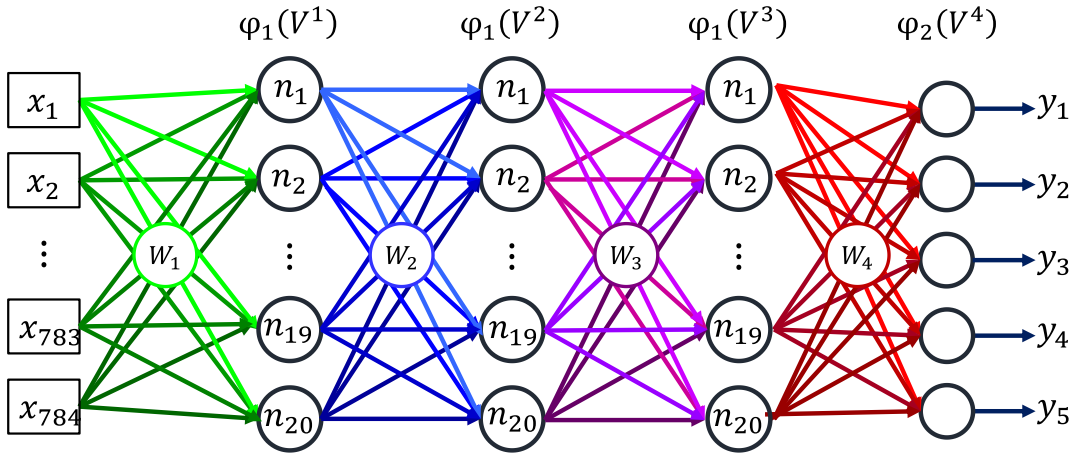

Fig. 4.3. Arquitectura de la red neuronal con aprendizaje profundo para el reconocimiento de dígitos decimales.

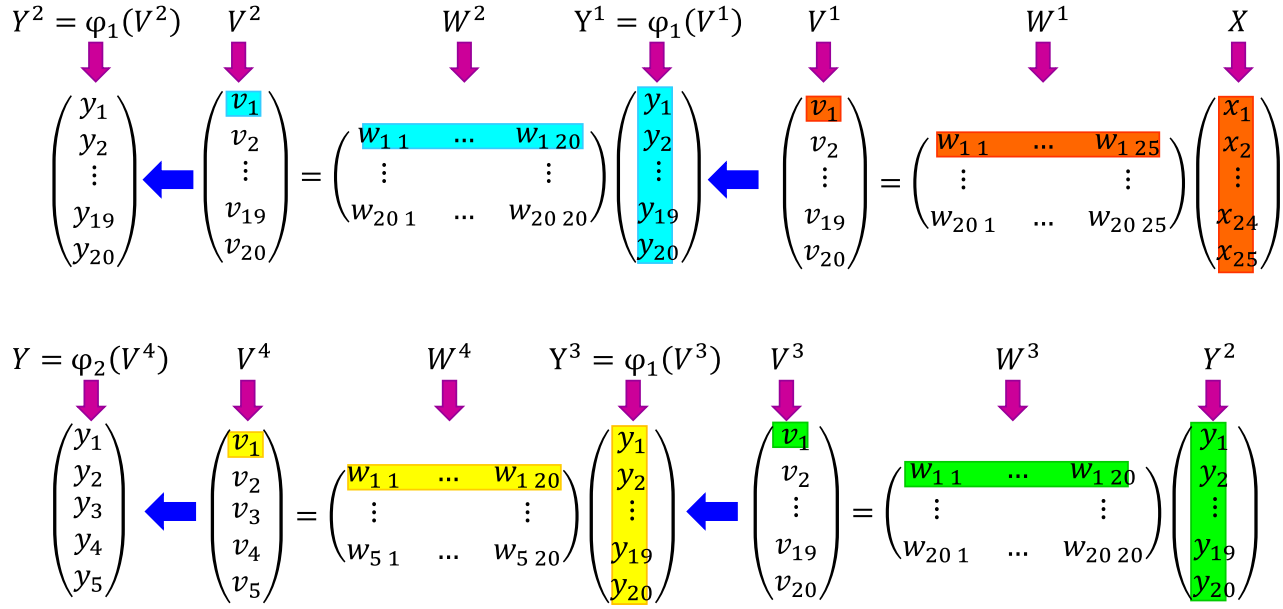

Fig. 4.4. Dimensiones de las matrices de pesos utilizadas para la red entrenada con las ideas del aprendizaje profundo. La función de activación φ_1 es la función ReLU y la función φ_2 es la función Softmax.

Para el entrenamiento de esta red se utilizó el siguiente pseudocódigo

i. *Para todos los datos de entrenamiento:*
ii. *Convertir la matriz de dato de entrenamiento en un vector columna (25x1).*
iii. *Obtener la suma ponderada de la primera capa (W_1x)*
iv. *Utilizar la función ReLU en la suma ponderada del punto iii.*
v. *Realizar iii y iv para las capas 2 y 3*
vi. *Obtener la suma ponderada para la capa de salida y activarla con la función SoftMax*

vii. *Determinar el error de salida y calcular la delta correspondiente*

viii. *Aplicar el algoritmo de propagación inversa para determinar los errores de las capas ocultas 3, 2 y 1*

ix. *Calcular el producto de la derivada de la función ReLU sobre las sumas ponderadas de las matrices 3, 2 y 1 y los errores correspondientes del punto viii, para obtener las deltas respectivas* ($\delta^n = \varphi'(v^n)e^n$)

x. *Actualizar las matrices de pesos con* $\Delta W^n = \alpha\delta^n Y^{nT}$

xi. *Repetir los pasos i-ix hasta alcanzar el número de épocas o el error deseado.*

A continuación, se muestran los resultados del entrenamiento de esta red.

4.5 RESULTADOS DE LA RED NEURONAL CON APRENDIZAJE PROFUNDO

En la Fig. 4.3 se presenta la arquitectura de la red neuronal que se utilizara para implementar las ideas de aprendizaje profundo. Esta red consta de una capa de entrada con 25 nodos, dos capas ocultas con 20 nodos y una capa de salida con 5 nodos

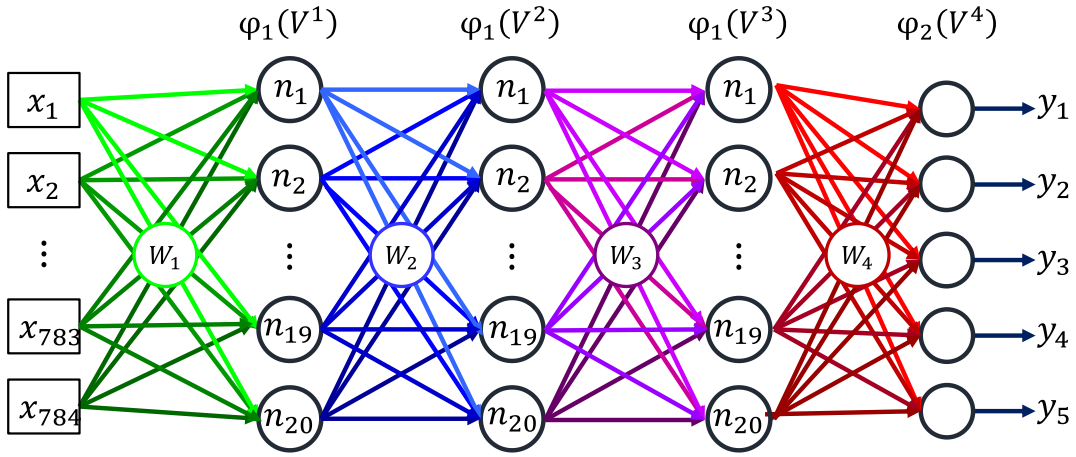

Fig. 4.5. Arquitectura de la red neuronal con aprendizaje profundo para el reconocimiento de dígitos decimales.

En la Fig. 4.6 se muestran los resultados de le evolución del error en los nodos de salida a medida que avanza el entrenamiento de la red. Como puede observarse en este caso el entrenamiento es mucho más eficiente puesto que el error decae rápidamente comparado con los resultados obtenidos con la SGD. En Fig. 4.7 la se muestra la GUI que vimos en la sección anterior, pero en esta se utiliza la función de activación ReLU.

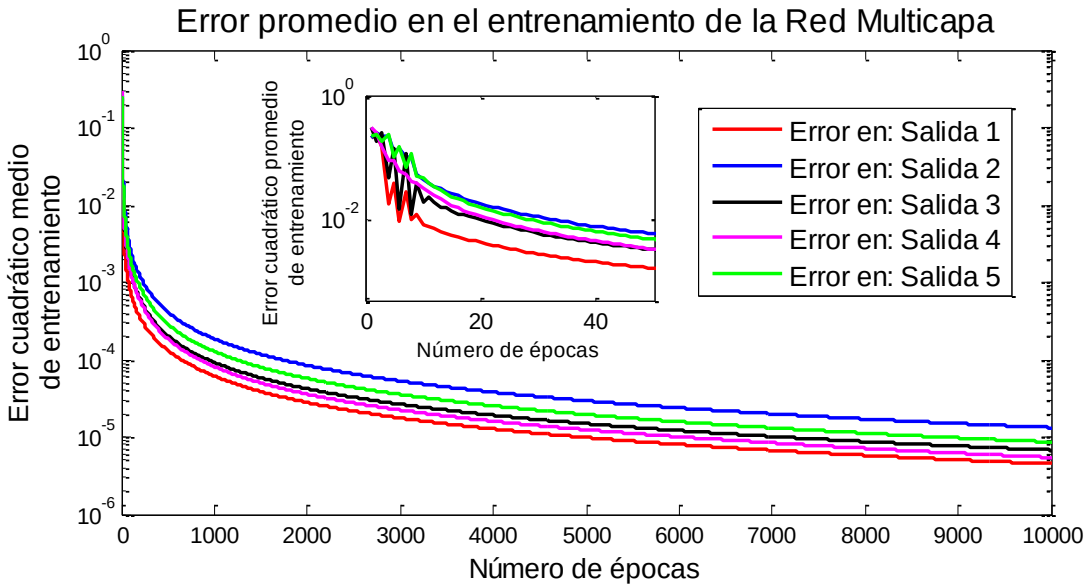

Fig. 4.6. Evolución del error cuadrático promedio en el entrenamiento de la red neuronal utilizando el aprendizaje profundo. Los errores en las salidas de la red neuronal cambian ahora muy rápidamente, debido a esto, se utiliza escala logarítmica en el eje vertical.

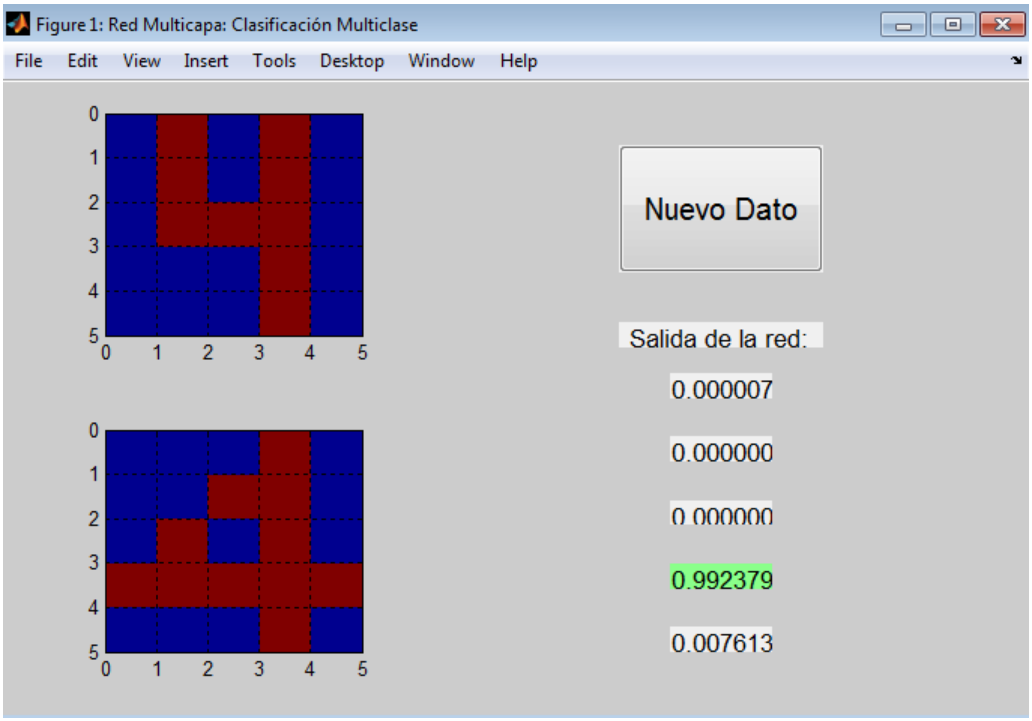

Fig. 4.7. GUI utilizada para el reconocimiento de dígitos utilizando una red neuronal multiclase con una función de activación SoftMax. La red neuronal identifica el dato introducido como el digito "4" con una certeza del 99% mientras que en la Fig. 3.7 lo hace con una certeza del 89%.

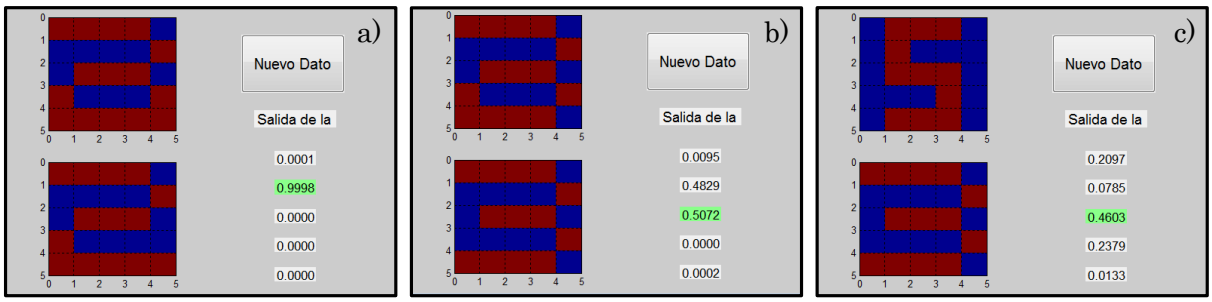

Fig. 4.8. Resultados de la red neuronal profunda. En este caso se utilizan los mismos ejemplos que los mostrados en la Fig. 3.8. Aun cuando el error en el entrenamiento es mucho menor en esta red que la mostrada en el capítulo 3 pareciera que su desempeño no es particularmente mejor, sin embargo, esto podría deberse no a la eficacia de la red sino quizá al limitado número de datos de entrenamiento.

En el recuadro puede observarse que este proceso de entrenamiento genera disminuciones en el error por cerca de tres órdenes de magnitud durante las primeras 50 épocas de entrenamiento. A pesar del éxito de esta red si se quisiese utilizar, rápidamente descubriríamos que su desempeño con datos reales no sería el esperado. Esto se debe básicamente a que hemos utilizado datos de entrenamiento muy idealizados (dígitos escritos en imágenes muy claras y prácticamente libres de ruido). Cuando se requiere trabajar con datos reales es necesario incorporar nuevas capas que nos permitan extraer información relevante de las imágenes que deseamos procesar, a continuación, se describen las redes ConvNet que resuelven este tipo de problemas.

Capítulo 5. Red Neuronal Convolucional

Este capítulo presenta la Red Neuronal Convolucional (ConvNet, por sus siglas en inglés), que es una aplicación de la clasificación. Esta es una Red Neuronal Profunda que se compone por dos partes. En este ejemplo se puede ver fácilmente la importancia de la mejora y aumento en las capas ocultas, así como los métodos para optimizar el rendimiento de la red neuronal.

ConvNet es una Red Profunda que imita cómo la corteza visual del cerebro procesa y reconoce imágenes. Para el reconocimiento de patrones en imágenes no se procesa directamente la imagen original, se tiene primero que resaltar algunos detalles significativos, es por esto que ConvNet está formada por dos partes: una red neuronal que extrae las características de la imagen original y una red neuronal de clasificación multiclase. A continuación, se describen ambas capas.

5.1 Extractor de características

La Red Neuronal que resalta las características está formada por múltiples capas de convolución [8] y agrupación. Esta capa se puede considerar como un grupo de filtros digitales que extraen la información relevante y reducen además el tamaño de la imagen original lo que minimiza la carga computacional [4, 7, 9]. El proceso realizado en la capa de convolución se muestra esquemáticamente en la Fig. 5.1.

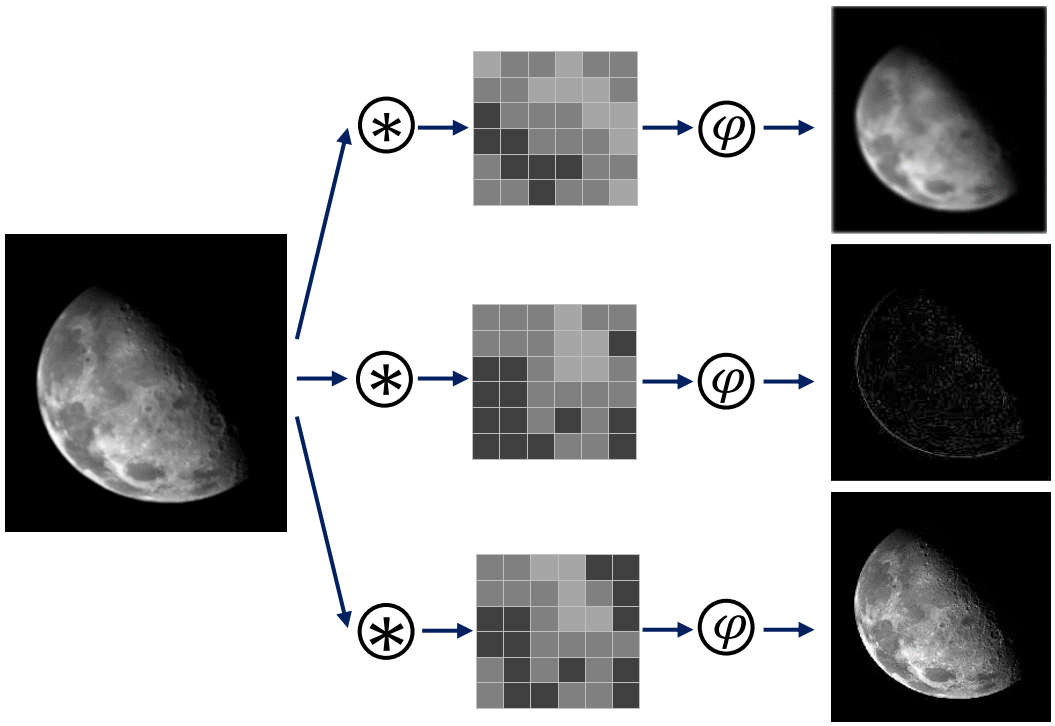

*Fig. 5.1. La capa de convolución realiza la operación de convolución entre el filtro y la imagen a procesar. Los * dentro de los círculos denotan la operación convolución, φ es la función de activación y las matrices cuadradas en escala de grises son los filtros de convolución.*

5.1 PROCESO DE CONVOLUCIÓN

Los filtros de la capa de convolución son matrices bidimensionales cuadradas, suelen ser de 5x5 o de 3x3 en escala de grises. En el proceso de convolución, el valor de los pixeles de entrada alrededor de un píxel, afectan la salida de este. Por ejemplo, supongamos que tenemos una imagen y un filtro de la siguiente forma:

$$I = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 4 & 4 & 6 \\ 7 & 8 & 4 & 9 \end{bmatrix} \qquad F = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

Fig. 5.2. Matriz imagen: I, filtro de convolución: F.

La convolución entre estas dos matrices de puede representar como se muestra en la Fig. 5.3.

$$A = F \circledast I = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \circledast \begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 4 & 4 & 6 \\ 7 & 8 & 4 & 9 \end{bmatrix}$$

Fig. 5.3. Convolución entre el filtro F y la matriz I.

Para realizar la convolución primero se debe "rotar" el filtro por 180º denotemos a esta nueva matriz como *G*. El proceso de convolución para la primera entrada de A se realiza sumando el producto de cada entrada de la matriz G por cada elemento de una submatriz de *I* con las dimensiones de *G* tal y como se ve en la Fig. 5.4.

$$a_{11} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} = 1 * 1 + 2 * 1 + 3 * 1 + 4 * 2 = 14$$

Fig. 5.4. La primera entrada de la matriz A es el resultado la suma del producto término a término de la matriz F y la primera submatriz con el tamaño de F de la matriz I.

Este proceso se repite de izquierda a derecha desde a_{11} hasta llegar a a_{13} y en seguida se repiten estas acciones para cada fila, entonces tenemos la matriz *A* como se muestra en la Fig. 5.5.

$$A = \begin{bmatrix} 1*1+2*1+ \\ 3*1+4*2 & 1*1+2*2+ \\ 3*2+4*3 & 1*2+2*3+ \\ 3*3+4*4 \\ 1*1+2*2+ \\ 3*2+4*4 & 1*2+2*3+ \\ 3*4+4*4 & 1*3+2*4+ \\ 3*4+4*4 \\ 1*2+2*4+ \\ 3*7+4*8 & 1*4+2*4+ \\ 3*8+4*4 & 1*4+2*6+ \\ 3*4+4*9 \end{bmatrix} = \begin{bmatrix} 14 & 23 & 33 \\ 27 & 36 & 47 \\ 63 & 52 & 64 \end{bmatrix}$$

Fig. 5.5. La i-esima entrada de la matriz A es la sumatoria del producto término a término de la matriz F con la i-esima submatriz de la matriz I.

El proceso descrito anteriormente da como resultado una matriz más pequeña, sin embargo, en realidad el proceso de convolución debe dar como resultado una matriz de mayores dimensiones esto debido a que la operación convolución en realidad requiere un desplazamiento desde $-\infty$ hasta ∞ (en ambos ejes) de la matriz del filtro. Usualmente para realizar esta operación se deben agregar ceros a la matriz del filtro, sin embargo, esto se ha omitido por sencillez. La capa de convolución repite este proceso para cada filtro y entrega un grupo de imágenes y a estas se les aplica la función de activación para esto se puede ocupar la función ReLU, la función Sigmoide o la función de tangente hiperbólica tanh (x).

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5.1}$$

5.2 CAPA DE AGRUPACIÓN

La capa de agrupación reduce el tamaño de la imagen ya que combina los pixeles vecinos de un píxel de entrada y entrega un solo píxel de salida, para esto hay que determinar la forma en que se relacionaran estos pixeles, los pixeles combinados generalmente se seleccionan de una submatriz cuadrada, el tamaño de esta depende del problema a tratar. El valor representativo generalmente es se establece como el máximo o el promedio de estos. Como ejemplo supongamos que tenemos la siguiente matriz:

1	1	2	3
0	2	3	4
2	3	4	6
7	8	5	9

Dividiremos la matriz original en submatrices de 2x2, combinaremos estos pixeles y generaremos una matriz de 2x2 como se muestra en la Fig. 5.6

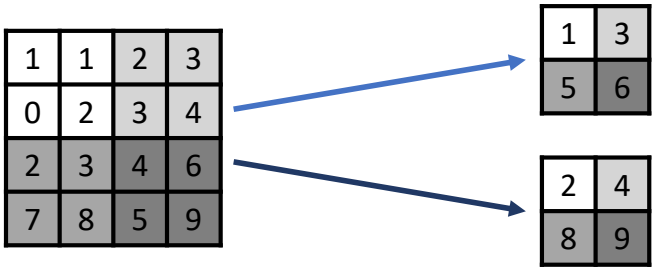

Fig. 5.6. La matriz original se reduce a una matriz de 2x2, la matriz superior se genera promediando los pixeles y la inferior con el máximo valor de entradas sombreadas con el mismo color.

La reducción de la imagen disminuye la carga computacional. A continuación, vamos a mostrar un ejemplo de aplicación de la red convolucional

5.3 IMPLEMENTACIÓN Y ANÁLISIS DE RESULTADOS APLICADOS AL RECONOCIMIENTO DE IMÁGENES

En esta sección se muestra una aplicación de la Red Neuronal Convolucional que procesa imágenes de dígitos escritos a mano y los clasifica según su estructura.

Este ejemplo ocupa como datos de entrenamiento una base de imágenes que contiene 70 000 fotografías de dígitos escritos a mano. Cada imagen tiene una resolución de 28 x 28 pixeles y pueden ser obtenidos de la base de datos de MNIST (Modified National Institute of Standards and Technology database) [4, 10].

Utilizaremos 10 000 imágenes, de estas 8 000 serán utilizadas como datos para el entrenamiento y las 2 000 restantes serán usadas para la validación. La Fig. 5.7 muestra un conjunto de estos datos.

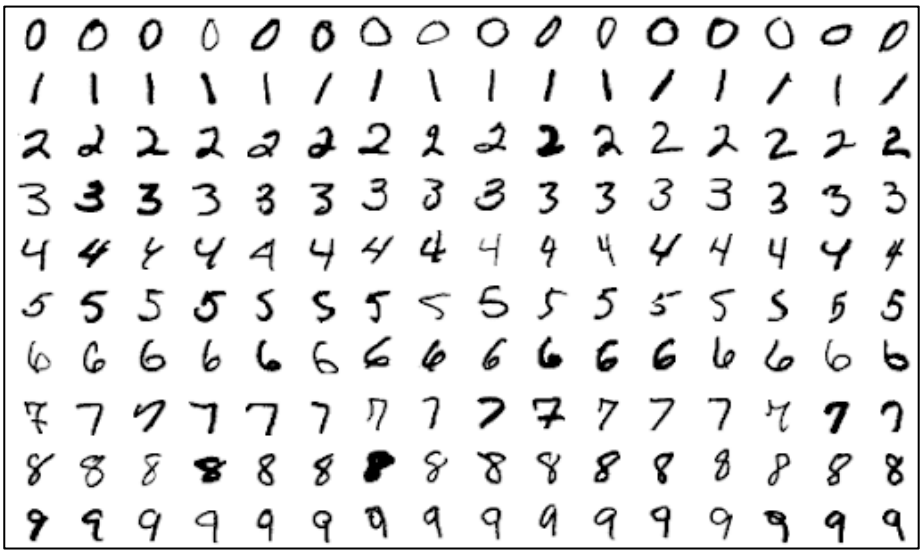

Fig. 5.7. Los datos de entrenamiento para este ejemplo es una base de datos que contiene imágenes de dígitos escritos a mano con una resolución de 28 x 28.

La red neuronal de este ejemplo contendrá 784 nodos de entrada ya que tenemos imágenes de 28 x 28 y una sola capa de convolución con 20 filtros de 9 x 9 pixeles. La salida de esta capa pasara por la función ReLU y después pasara por una capa de filtros de agrupación para dar como resultado, matrices de 10 x 10. La red para la clasificación cuenta con una capa oculta con 2000 nodos que usan como función de activación la función ReLU. Finalmente, la capa de salida cuenta con 10 nodos debido a que se tienen 10 grupos de clasificación. En esta última capa se usa la función de activación SoftMax. La Fig. 5.8 muestra la red neuronal convolucional y la Fig. 5.9 muestra la red neuronal de clasificación.

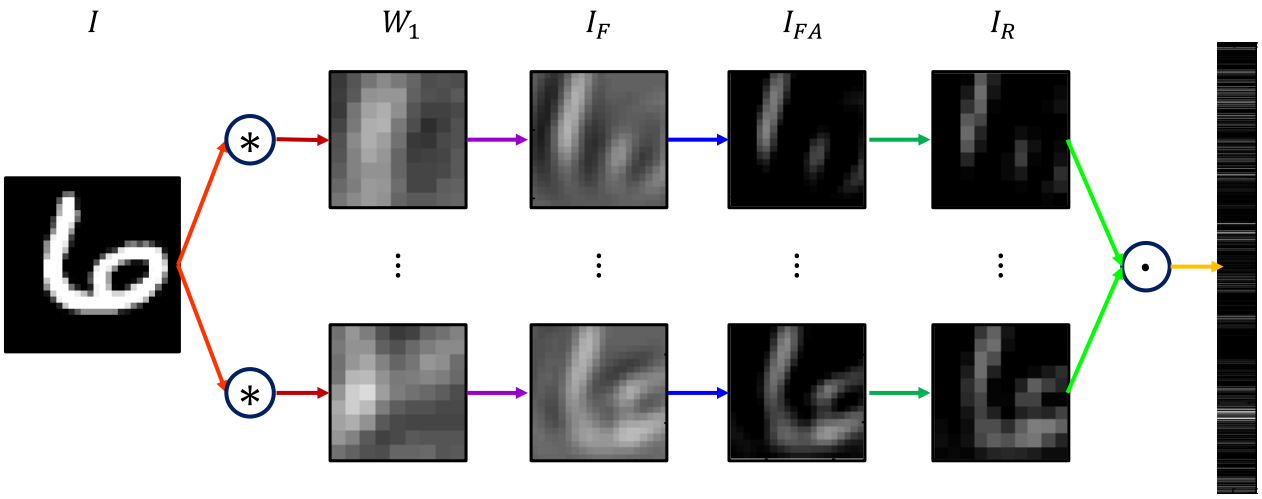

Fig. 5.8. La Red Neuronal convolucional tiene 20 filtros de 9 x 9. El resultado de esta capa son 20 matrices de 20 x 20. la capa de agrupación promedia cada una de las 20 imágenes y se toman solo los pixeles impares por lo que el resultado es un conjunto de 20 matrices de 10 x 10. Finalmente se combinan estas 20 matrices en un solo vector columna de 2000 x 1.

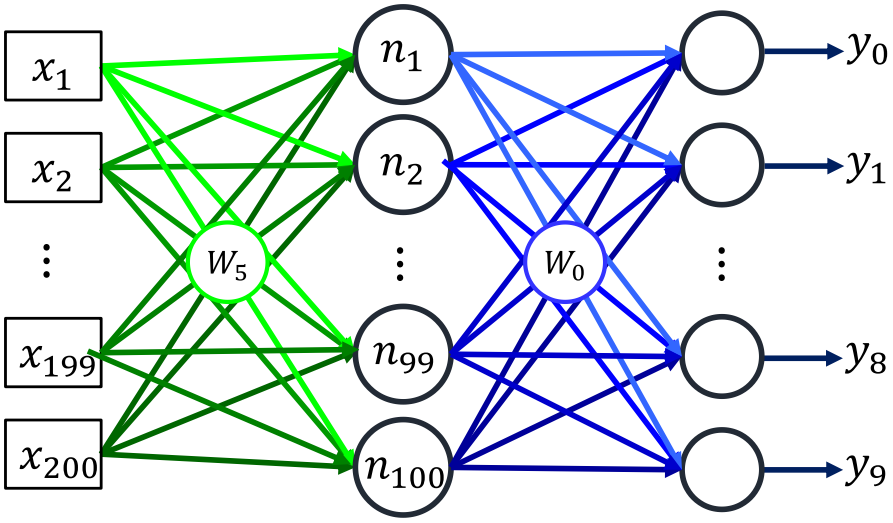

Fig. 5.9. La Red Neuronal de Clasificación toma como entrada la salida de la red neuronal convolucional, tiene una capa oculta con 100 nodos y la capa de salida tiene 10 nodos.

Como se observa de la Fig. 5.9 la red de clasificación consta de dos mil nodos de entrada, una matriz W_5 de 100 x 2000 cuya salida es activada con la función ReLU y cuyos resultados son pesados por la capa de salida de la matriz W_o de 10 x 100 cuyo resultado es finalmente activado con la función SoftMax para tener de esta manera el vector de salida de 10 x 1. De esta manera el pseudoalgoritmo de entrenamiento para esta red neuronal es el siguiente:

i. *Para todos los datos de entrenamiento realizar lo siguiente:*
ii. *Para un grupo de 100 imágenes realizar lo siguiente:*
iii. *Tomar una imagen del digito escrito y aplicar los 20 filtros de convolución.*

iv. *Activar las imágenes filtradas del punto anterior.*

v. *Aplicar un filtro promediador a cada una de las imágenes activadas y seleccionar solo los pixeles impares para generar una reducción en las imágenes.*

vi. *Combinar las imágenes del punto anterior para crear un vector de 2000 x 1.*

vii. *Determinar la suma ponderada con una matriz de pesos 100 x 2000.*

viii. *Activar el resultado del punto anterior con la función ReLU.*

ix. *Determinar la suma ponderada del resultado anterior con la matriz W_0 de 10 x 100.*

x. *Obtener la salida mediante la activación con la función SoftMax con los resultados del grupo anterior.*

xi. *Aplicar el algoritmo de propagación inversa para determinar los errores y las deltas en cada capa.*

xii. *Una vez realizados los puntos ii-ix calcular el promedio de todas las deltas.*

xiii. *Actualizamos los coeficientes de las matrices de convolución y de pesos W_0 y W_5.*

xiv. *Repetir los puntos ii-xiii hasta cubrir todos los datos de entrenamiento.*

A continuación, se muestran los resultados de esta red convolucional.

5.4 RESULTADOS DEL ENTRENAMIENTO DE UNA RED CONVOLUCIONAL PARA LA CLASIFICACIÓN DE DÍGITOS ESCRITOS A MANO

Después de haber entrenado la red neuronal durante tres épocas (8000 imágenes por cada época) se obtuvieron los siguientes resultados.

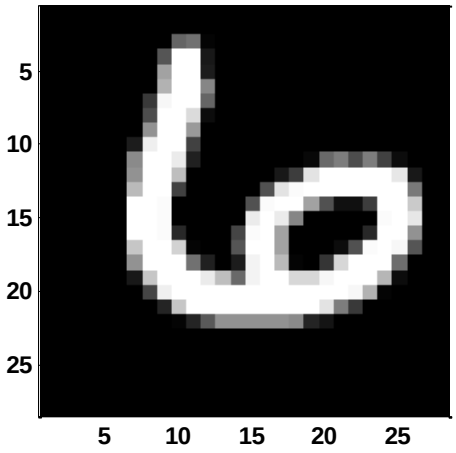

Fig. 5.10. Imagen de entrada del digito decimal 6escrito a mano extraída de la base de datos MNIST.

En la Fig. 5.10 se muestra un ejemplo de las fotografías que se utilizaron como datos de entrenamiento. Como puede verse esta imagen corresponde al digito 6. Después

de haber entrenado la red los filtros de convolución presentan la apariencia mostrada en la Fig. 5.11.

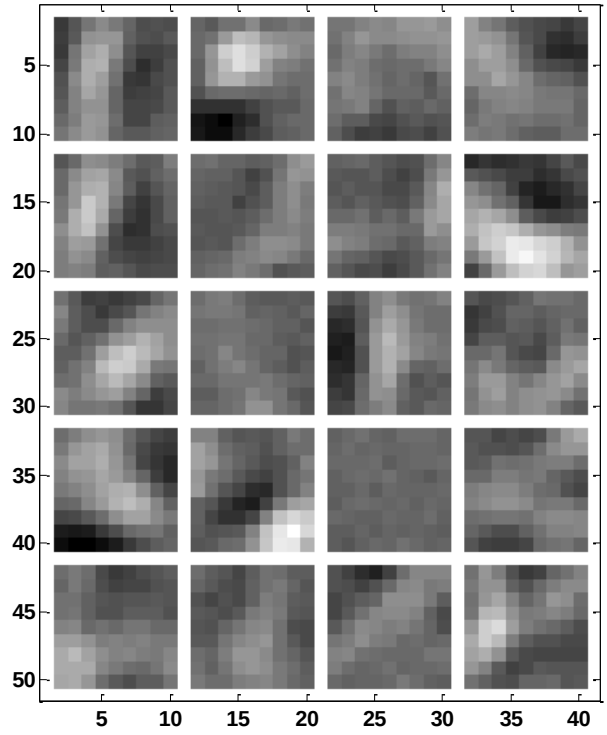

Fig. 5.11. Imagen de las 20 matrices utilizadas como filtros de convolución para obtener los mapas de características de las imágenes de los dígitos escritos a mano.

Las imágenes filtradas obtenidas utilizando los filtros de convolución de la Fig. 5.11 se muestran en la Fig. 5.12. Como se mencionó anteriormente, a estas imágenes se les conoce como los mapas de características. De la Fig. 5.12 puede observarse que estos mapas lo que muestran en general es una especie de "trazos básicos" que están presentes en el digito analizado. Por ejemplo, la imagen (1,1), renglón/columna, puede pensarse que en esencia está compuesta por dos trazos rectos: uno largo a la izquierda y uno pequeño a la derecha. Algo similar ocurre en la imagen (1,2) pero ahora los trazos son horizontales. La imagen (2,4) muestra una especie de segmentos curvos entrelazados. Puede decirse entonces que estos filtros convolucionales descomponen la imagen en estos trazos básicos y esta información es utilizada más adelante para determinar el digito que está presente en la imagen.

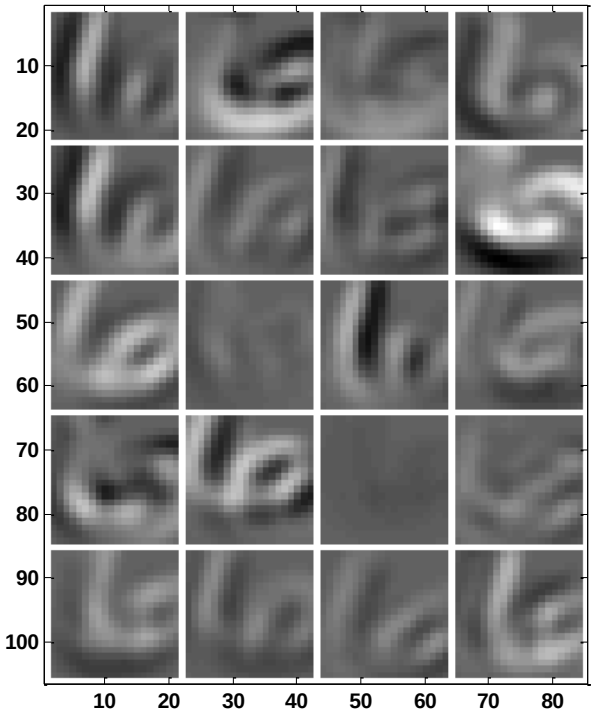

Fig. 5.12. Resultado de los 20 filtros aplicados a la imagen de entrada de la Fig. 5.10.

Por otra parte, en la Fig. 5.13 se muestran los resultados de la activación de estos mapas de características. Podemos decir que en este caso la activación es una manera de "realzar" las características que ya se perciben desde lo mapas de la Fig. 5.12. Por ejemplo, en la imagen (1,1) se ve aquí más claro que está formada básicamente por dos trazos verticales mientras que en la imagen (2,4) es más evidente la presencia de dos trazos curvos. Finalmente, en la Fig. 5.14 se muestran los resultados de la capa de reducción (polling en inglés). En este caso esta etapa de la red utilizó un filtro de convolución de 2x2 para calcular el promedio en la imagen. Después de esto, de la imagen promediada se ha tomado un pixel y se ha desechado el siguiente, este proceso se repitió tanto en los renglones como en las columnas y el resultado ha sido 20 imágenes reducidas en tamaño a 10x10 pixeles. Como ya se ha mencionado, esto se hace para minimizar la carga computacional.

Una vez que se tienen estas 20 imágenes reducidas se crea un vector columna con la información de todas estas imágenes dando como resultado un vector columna de 2000x1 pixeles, el cual es utilizado como información de entrada para la red de clasificación que está compuesta por dos matrices: la capa oculta de 100x2000 y la capa de salida que es de 10x100.

Un aspecto que es muy interesante de estas redes neuronales es el hecho de que una vez entrenadas, desde nuestro punto de vista, la información relevante se encuentra en las matrices de los filtros de convolución y en las matrices de la red de clasificación (capa oculta y la de salida). Sin embargo, si uno observa los valores de estas matrices, parecieran ser valores sin sentido tal y como se observa en las Fig. 5.11 y Fig. 5.16.

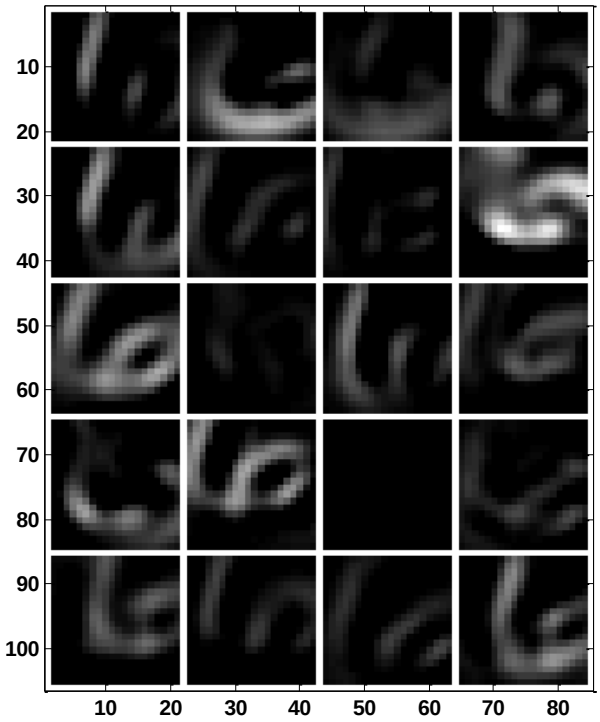

Fig. 5.13. Resultado de la activación mediante la función ReLU en las imágenes de la Fig. 5.12.

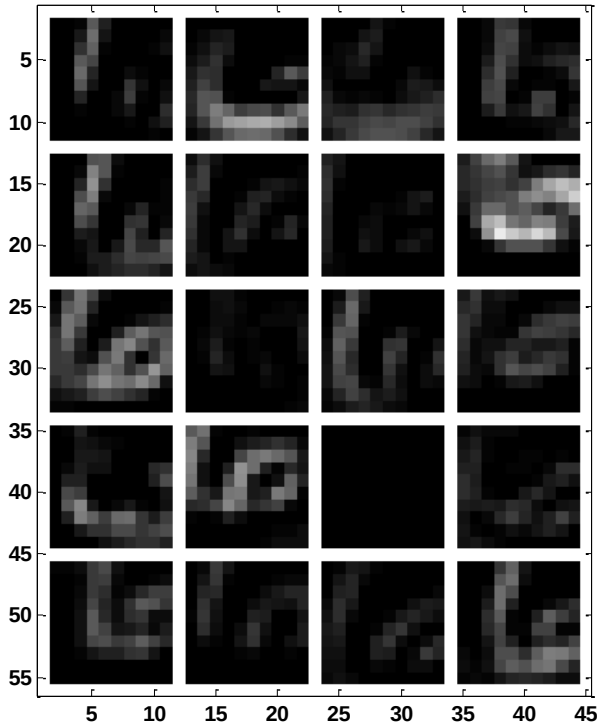

Fig. 5.14. Resultado de la reducción de las imágenes de la Fig. 5.13

Fig. 5.15. Reagrupación de los datos de la Fig. 5.14 que a su vez son los datos de entrada de la red neuronal de clasificación.

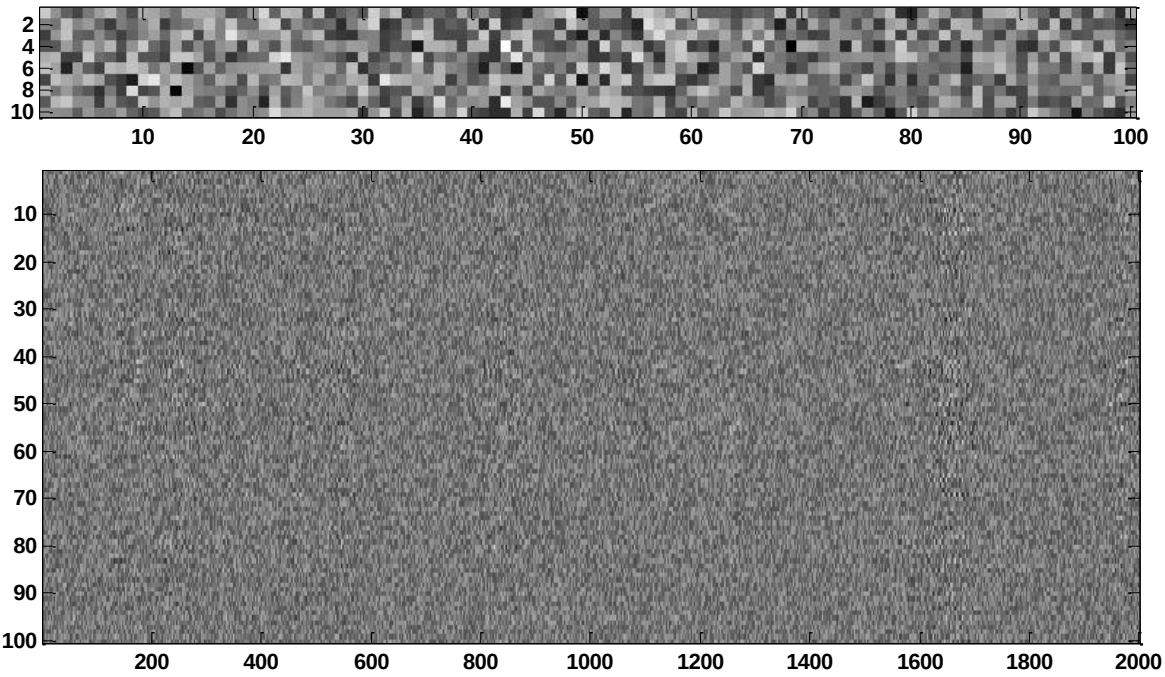

Fig. 5.16. Matriz de pesos de la red de clasificación: a) capa oculta y b) capa de salida.

Como puede observarse aun cuando la información de esta red de clasificación recae fuertemente en las matrices de la red de clasificación, su contenido presenta un "aspecto" que les hace lucir como información plagada de señales más bien aleatorias que aquellas que a primera vista uno podría pensar que debiese existir en estas matrices.

A pesar de esta impresión es importante recordar que si pudiésemos asignar algún valor de importancia dentro de las componentes de la red son justamente estas matrices los elementos más valiosos en el desempeño de este tipo de sistemas.

Como se había mencionado una vez que se ha entrenado la red con 8 000 de las 10 000 imágenes disponibles pueden utilizarse las 2 000 imágenes restantes para determinar la eficiencia del reconocimiento de esta red. El resultado de esta validación dio como resultado que el 93.5% de las imágenes analizadas es un resultado correcto. De esta manera podemos decir en términos generales, que la red funciona muy bien.

Capítulo 6. Conclusiones

En este trabajo se han presentado conceptos básicos acerca del Aprendizaje de Máquina, Aprendizaje Profundo y las Redes Convolucionales. Basados en los resultados obtenidos con los distintos programas desarrollados a lo largo del trabajo podemos llegar a las siguientes conclusiones:

- Una red neuronal compuesta por una sola capa puede resolver solamente problemas muy sencillos por lo que su uso en aplicaciones reales es prácticamente nulo.
- Aplicaciones reales con un mayor grado de complejidad requieren necesariamente el uso de redes con capas ocultas.
- El uso de Gradiente descendiente estocástico con función de activación sigmoide es una técnica que ofrece un entrenamiento de la red con un cambio en el error de entrenamiento que puede ser observado en escala lineal (el decaimiento del error es "lento").
- Una mejora dramática en el comportamiento del decaimiento del error de entrenamiento se da cuando se utiliza la entropía cruzada para estimar la regla delta en la corrección de los términos de peso de las marices que conforman las distintas capas de la red.
- El uso de redes convolucionales para tares de clasificación ofrecen resultados sorprendentes que pueden ser implementados de manera rápida y simple en cualquier lenguaje de programación.
- Particularmente la red implementada para el reconocimiento de dígitos escritos a mano tiene una eficacia de reconocimiento de tales dígitos del orden de 93%.
- Aun cuando esta red convolucional pudiera parecer muy eficiente debe notarse que su entrenamiento aún no resulta ser el mejor. Por ejemplo, la red se entrenó con imágenes en escala de grises, por lo que esta imposibilitada para reconocer dígitos a color. Por otra parte, esta red solo se entrenó con dígitos escritos a mano lo que significa que probablemente será incapaz de reconocer dígitos que estén expresados de manera distinta. Por este motivo si estuviésemos interesados en ampliar las capacidades de dicha red será necesario elegir un conjunto de datos de entrenamiento más amplio o aplicar las ideas del llamado "Aumento de Datos" (data augmentation en inglés).

BIBLIOGRAFÍA

1. "Matlab deep learning", Phil Kim, Ed. Apress, 2017.
2. "Machine learning. A probabilistic perspective", Kevin P. Murphy. The MIT press.
3. "Understanding machine learning from theory to algorithms". Shai Shalev – Shwartz and Shai Ben – David. Cambridge university press.
4. "Hands-on machine learning with Scikit-Learn, Keras & TensorFlow: Concepts, tools, and techniques to build intelligent systems", Aurélien Géron, Ed. Oreilly Media, 2nd Ed., 2019.
5. "MATLAB Machine Learning Recipes, A Problem-Solution Approach", Michael Paluszek, Stephanie Thomas, Ed. Apress, 2019.
6. "Data-driven science and engineering: Machine learning, dynamical systems, and control", Steven L. Bruton, J. Nathan Kutz, Ed. Cambridge University Press, 1rst Ed., 2019.
7. "Visión por computador: Imágenes digitales y aplicaciones", Gonzalo Pajares Martinsanz, Jesús M. de la Cruz García, Ed. Alfaomega Ra-Ma 2ª Ed, 2008.
8. "Digital Image processing", Rafael C. González, Richard E. Woods. Ed. Pearson Prentice Hall, 3rd Ed., 2008.
9. "Practical Machine Learning and Image Processing: For Facial Recognition, Object Detection, and Pattern Recognition Using Python", Himanshu Singh, Ed. Apress, 2019.
10. Base de datos: https://deepai.org/dataset/mnist.