PROYECTO FIN DE CARRERA

APLICACIONES DE LAS REDES NEURONALES EN EL CAMPO DE LAS TELECOMUNICACIONES

Autor: Ángel Rubio Carta

Tutor: Carlos Crespo Cadenas

Ingeniero de Telecomunicación
Escuela Superior de Ingenieros
Universidad de Sevilla

Sevilla, Julio del 2002

ÍNDICE

1 INTRODUCCIÓN A LAS REDES NEURONALES 5

1.1 ¿QUÉ ES UNA RED NEURONAL? 5

1.2 MODELOS DE UNA NEURONA 7

1.2.1 Tipos de funciones de activación 8

1.3 ARQUITECTURA DE REDES 9

1.3.1 Redes monocapa 10

1.3.2 Redes multicapa 10

1.3.3 Redes recurrentes 11

1.3.4 Estructuras Lattice 12

1.4 REPRESENTACIÓN DEL CONOCIMIENTO 12

1.4.1 Reglas para la representación del conocimiento 13

1.4.2 Cómo incluir información a priori en el diseño de una red neuronal 14

1.4.3 Cómo incluir invarianzas en el diseño de una red neuronal 14

1.5 INTELIGENCIA ARTIFICIAL Y REDES NEURONALES 15

2 PROCESOS DE APRENDIZAJE 17

2.1 ALGORITMOS DE APRENDIZAJE 18

2.1.1 Aprendizaje por corrección de error 18

2.1.2 Aprendizaje Hebbiano 20

2.1.3 Aprendizaje competitivo 22

2.1.4 Aprendizaje de Boltzmann 24

2.2 PARADIGMAS DE APRENDIZAJE 25

2.2.1 Aprendizaje supervisado 25

2.2.2 Aprendizaje reforzado 27

2.2.3 Aprendizaje no supervisado o auto-organizado 27

2.2.4 Comparativa entre aprendizaje supervisado, reforzado y no supervisado 28

2.3 TAREAS DE APRENDIZAJE 29

2.4 ADAPTACIÓN Y APRENDIZAJE 30

3 EL PERCEPTRÓN 32

3.1 CONSIDERACIONES BÁSICAS 33

3.2 TEOREMA DE LA CONVERGENCIA DEL PERCEPTRÓN 34

3.2.1 Resumen del algoritmo de convergencia del perceptrón 39

3.3 MEDIDA DEL RENDIMIENTO 41

4 PERCEPTRONES MULTICAPA 43

4.1 INTRODUCCIÓN 43
4.2 ALGORITMO BACKPROPAGATION 44
4.2.1 Caso I: La neurona j es un nodo de salida 47
4.2.2 Caso II: La neurona j es un nodo oculto 48
4.2.3 Las dos pasadas del cálculo 50
4.2.4 No linealidad sigmoidal 51
4.2.5 Tasa de aprendizaje 52
4.2.6 Modos de entrenamiento 54
4.2.7 Criterio de parada 55
4.3 INICIALIZACIÓN 56
4.4 APROXIMACIÓN DE FUNCIONES 58
4.4.1 Teorema de la Aproximación Universal 58
4.4.2 Consideraciones prácticas 59

5 APLICACIÓN DE LAS REDES NEURONALES PARA LA APROXIMACIÓN DE FUNCIONES CARACTERÍSTICAS DE UN TRANSISTOR HEMT 60

5.1 APROXIMACIÓN DE LA CURVA I_{DS} FRENTE A V_{GS} 61
5.2 APROXIMACIÓN DE LA CURVA G_M FRENTE A V_{GS} 71
5.3 APROXIMACIÓN DE LA CURVA I_{DS} FRENTE A V_{GS}, V_{DS} (MODELO GSR1) 86
5.4 APROXIMACIÓN DE LA CURVA I_{DS} FRENTE A V_{GS}, V_{DS} (MODELO DE ANGELOV) 94
5.5 APROXIMACIÓN DE LA CURVA G_{DS} FRENTE A V_{GS}, V_{DS} (MODELO DE FUJII) 99
5.6 EVALUACIÓN DE LOS RESULTADOS OBTENIDOS 103
5.6.1 Aproximaciones de I_{ds} y g_m. Derivación e integración 103
5.6.2 Aproximación de I_{ds} frente a V_{gs} y V_{ds} 110
5.6.3 Aproximación de G_{ds} frente a V_{gs} y V_{ds} 111
5.6.4 Conclusiones 111

6 APLICACIÓN DE LAS REDES NEURONALES PARA LA PREDICCIÓN DE LAS PÉRDIDAS BÁSICAS DE POPAGACIÓN 112

6.1 FUNDAMENTOS 112
6.1.1 Método de Okumura-Hata 113
6.1.2 Método COST 231 114
6.2 APLICACIÓN 114

7 APLICACIÓN DE LAS REDES NEURONALES PARA LA COMPRESIÓN DE IMÁGENES DE ESCALA DE GRISES 118

7.1 FUNDAMENTOS 118
7.2 APLICACIÓN 119
7.2.1 Empleando 32 neuronas en la capa oculta 120
7.2.2 Empleando 16 neuronas en la capa oculta 121

7.2.3 Empleando 8 neuronas en la capa oculta..... 121

8 LÍNEAS DE AVANCE..... 122

8.1 COMPRESIÓN DE IMÁGENES..... 122

8.2 RECONOCIMIENTO DE DÍGITOS ESCRITOS A MANO..... 122

8.3 SINTETIZADOR DE TEXTO ESCRITO..... 124

9 BIBLIOGRAFÍA..... 126

CONTENIDO

Desde su concepción, las redes neuronales han sido utilizadas con profusión en diversos campos de la Ingeniería con el fin de resolver problemas de difícil solución mediante métodos matemáticos clásicos. Como indica el nombre de este proyecto, en él se trata de estudiar diversas aplicaciones de las redes neuronales en el campo de las Telecomunicaciones. El proyecto se estructura en diez capítulos. Los cuatro primeros capítulos están dedicados a una introducción en los conceptos básicos de las redes neuronales, los mecanismos de aprendizaje que se llevan a cabo en éstas, el estudio del perceptrón como elemento fundamental constituyente de las redes neuronales y el estudio de los perceptrones multicapa, es decir, la creación de redes neuronales mediante la distribución de varios perceptrones en distintas capas. En el capítulo cinco, obtendremos diversos modelos para la caracterización de transistores HEMT; en el seis, veremos la posibilidad de predecir el nivel de señal recibido en un sistema de telefonía móvil en una región dada a partir de medidas experimentales; en el siete, veremos cómo se pueden utilizar las redes neuronales para la compresión de imágenes. Para finalizar, el proyecto concluye con unas líneas de avance para trabajos posteriores.

1 INTRODUCCIÓN A LAS REDES NEURONALES

1.1 ¿Qué es una red neuronal?

Las redes neuronales artificiales nacieron desde el reconocimiento de que el cerebro realiza cálculos de una manera completamente diferente a como hacen las computadoras digitales. Desde comienzos del siglo XX se ha desarrollado una gran labor de investigación con el fin de adquirir un mayor conocimiento acerca del funcionamiento del cerebro humano, labor en la que contribuyó notablemente el científico Ramón y Cajal, quien introdujo el concepto de neurona como estructura constituyente del cerebro.

Los ordenadores digitales realizados por el hombre están diseñados para ejecutar ciertas tareas que han sido formuladas con exactitud. En cambio, el cerebro humano es capaz de procesar tareas como la visión, el habla, la memoria, reconocimiento espacial y temporal de patrones y otras muchas más en presencia de ruido, siendo todas estas tareas de gran dificultad para ser llevadas a cabos por ordenadores convencionales.

¿Cómo es capaz el cerebro humano de llevar a cabo estas tareas, teniendo en cuenta que las neuronas son cinco o seis órdenes de magnitud más lentas que las puertas lógicas realizadas con silicio?. Una posible respuesta a esta pregunta es que el cerebro distribuye el cálculo a realizar entre billones de neuronas relativamente sencillas, entre las cuales existe una gran interconectividad y están constantemente mandando y recibiendo información. Se estima que en el cerebro humano existen aproximadamente 10 billones de neuronas y 60 trillones de conexiones sinápticas. El cerebro humano posee una característica que podríamos denominar plasticidad y que permite al sistema nervioso adaptarse al entorno; esta adaptación se consigue mediante dos mecanismos: creación de una nueva conexión sináptica entre neuronas y la modificación de una enlace sináptico ya existente. Además, se han descubierto numerosos mecanismos en el cerebro, cada uno de los cuales resuelve ciertos problemas relacionados con el procesado simultáneo. Algunos de estos mecanismos son la asociación, la generalización y la organización. Todo esto fue lo que llevó al estudio y desarrollo de neuronas artificiales y redes neuronales artificiales, redes de arquitecturas en paralelo basadas en elementos de cómputo muy simples y conectados por enlaces de pesos variables. Así, una red neuronal, vista como una máquina adaptativa, puede ser definida de la siguiente manera:

> *Una red neuronal es un procesador distribuido y con alta capacidad de cálculo en paralelo con una tendencia natural a almacenar conocimiento experimental y a hacer éste disponible para su uso. Se asemeja al cerebro en dos aspectos:*
>
> 1. *El conocimiento es adquirido por la red a través de un proceso de aprendizaje.*

2. *La fuerza de las conexiones entra las neuronas conocidos como pesos sinápticos son usadas para almacenar el conocimiento.*

El procedimiento usado para realizar el proceso de aprendizaje se llama algoritmo de aprendizaje, algoritmo que tiene la función de modificar los pesos sinápticos de la red neuronal de forma que se alcance un objetivo de diseño deseado.

Aunque las redes neuronales pueden servir como modelos para comprender mejor el funcionamiento del cerebro humano, éstas son muy interesantes para resolver ciertos problemas. Por todo lo mencionado antes, podemos decir que el poder de cálculo de las redes neuronales deriva de, primero, su estructura distribuida en paralelo y, segundo, su habilidad para aprender y generalizar; por generalización se entiende el que la red neuronal produzca salidas razonables ante entradas no encontradas durante el entrenamiento (aprendizaje). Estas dos habilidades para el procesado de la información permiten que las redes neuronales sean capaces de resolver problemas complejos que de otra manera serían intratables.

El uso de redes neuronales ofrece las siguientes propiedades y capacidades:

1. *No linealidad*. Una neurona es básicamente un dispositivo no lineal, por lo que las redes neuronales compuestas de numerosas neuronas conectadas entre sí serán también no lineales, por lo que podrán ser utilizadas para realizar operaciones de filtrado que son superiores a las capacidades de las técnicas de filtrado lineal convencionales.

2. *Mapeado entre entrada y salida*. Un popular paradigma de aprendizaje llamado aprendizaje supervisado consiste en la modificación de los pesos sinápticos de una red neuronal aplicando un conjunto de ejemplos de entrenamiento. Cada ejemplo consiste en una señal de entrada y la correspondiente salida deseada. A la red se le presenta un ejemplo cogido al azar del conjunto y los pesos sinápticos se modifican de forma que se minimice la diferencia entre la respuesta deseada y la respuesta actual de la red producida por la señal de entrada en concordancia con un apropiado criterio estadístico. El entrenamiento de la red se repite con tantos ejemplos haya hasta que la red alcanza un estado estable en el que ya no existen cambios apreciables en el valor de los pesos sinápticos. Debido a esta propiedad es el que las redes neuronales sean empleadas para el reconocimiento de patrones definiendo regiones no lineales en el espacio de representación.

3. *Adaptabilidad.* Las redes neuronales tienen una intrínseca capacidad para adaptar los pesos sinápticos a los cambios en el entorno. Una red neuronal puede ser diseñada incluso para cambiar sus pesos sinápticos en tiempo real, aunque no hay que olvidar que la adaptabilidad no siempre conduce a robustez.

4. *Respuesta indicativa.* En el contexto de la clasificación de patrones, una red neuronal puede ser diseñada para proporcionar información no solo sobre qué patrón seleccionar, sino también sobre el grado de incertidumbre en la decisión tomada, la cual puede ser utilizada para rechazar patrones ambiguos.

5. *Información contextual.* El conocimiento está representado por la estructura y estado de activación de la red neuronal. Cada neurona puede verse afectada por la actividad global de todas las demás neuronas de la red. Por tanto, la información contextual es tratada con naturalidad por una red neuronal.

6. *Tolerancia a fallos*. Una red neuronal, implementada físicamente, es intrínsecamente tolerante a fallos en el sentido de que si alguna neurona o sus conexiones se ven dañadas, el rendimiento no se verá seriamente afectado.

7. *Implementabilidad como VLSI.* Las redes neuronales pueden ser empleadas para realizar cálculos con rapidez, de ahí que estén indicadas para su implementación usando tecnología VLSI, pudiendo ser empleadas como una herramienta para aplicaciones en tiempo real relacionadas con el reconocimiento de patrones, procesamiento de señales y control.

8. *Uniformidad en el análisis y diseño*. Se emplea la misma notación en todos los ámbitos que impliquen aplicaciones de redes neuronales, lo que permite la compartición de teorías y algoritmos de aprendizaje en diferentes aplicaciones y el que se puedan construir redes modulares a partir de la integración de diversos módulos.

9. *Analogía neurobiológica*. El diseño de una red neuronal está motivado por analogía con el cerebro humano. Neurobiólogos miran a las redes neuronales artificiales como una herramienta para interpretar el funcionamiento del cerebro.

1.2 Modelos de una neurona

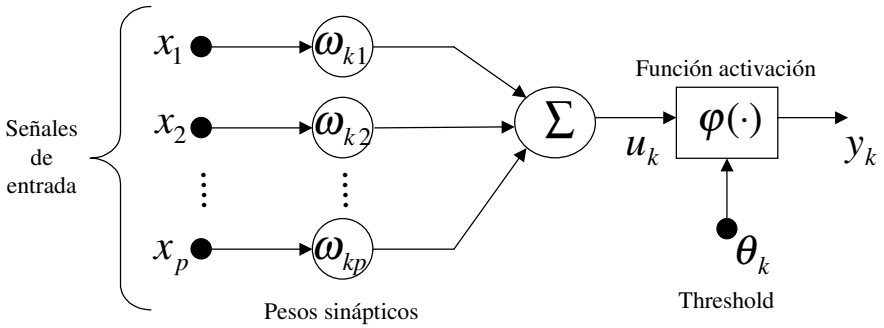
Una neurona es la unidad básica de procesado de información en una red neuronal. En el modelo de una neurona podemos identificar tres elementos básicos:

1. Conjunto de conexiones sinápticas, cada una de las cuales está caracterizada por un peso. Una señal x_j conectada a una neurona k se ve multiplicada por el peso sináptico w_{kj}.

2. Sumador para sumar las señales de entrada, multiplicadas por los respectivos pesos sinápticos, constituyendo una combinación lineal.

3. Función de activación para limitar la amplitud de la salida de la neurona.

En la siguiente figura se muestra el modelo de una neurona en el que también se ha aplicado un umbral aplicado externamente que tiene el efecto de disminuir la entrada de la función de activación:



Matemáticamente, podemos describir el comportamiento de una neurona mediante las siguientes ecuaciones:

$$u_k = \sum_{j=1}^{p} \omega_{kj} x_j$$

$$y_k = \varphi(u_k - \theta_k)$$

La señal u_k no es más que una combinación lineal de las señales de entrada, donde cada una de ellas es multiplicada por un valor denominado peso sináptico. La señal de salida de la neurona será una función, denominada función de activación, que depende de la combinación lineal de las señales de entrada y un umbral.

1.2.1 Tipos de funciones de activación

En este apartado vamos a ver las tres funciones de activación más básicas empleadas:

1. Función threshold. Viene dada por la siguiente expresión:

$$\varphi(v) = \begin{cases} 1 & si\ v \geq 0 \\ 0 & si\ v < 0 \end{cases}$$

Una neurona utilizando esta función es conocida como el modelo de McCulloch-Pitts y la salida en este caso es 1 cuando el nivel de actividad interna de la neurona, es decir, el valor de la combinación lineal de las señales de entrada menos el umbral, sea no negativo; valdrá cero en caso contrario.

2. Función Piecewise-Linear. Viene dada por la siguiente expresión:

$$\varphi(v)=\begin{cases} 1 & si\ v \geq 1/2 \\ v & si\ -1/2 < v < 1/2 \\ 0 & si\ v < -1/2 \end{cases}$$

Esta función de activación se puede ver como una aproximación a un amplificador no lineal, el cual presenta una región de funcionamiento lineal y dos zonas de saturación.

3. Función Sigmoide. Esta función de activación es la más empleada en la construcción de redes neuronales artificiales. Está definida como una función creciente que presenta propiedades asintóticas y de suavidad. Un ejemplo de función de activación sigmoide es la función logística, que viene dada por la siguiente expresión:

$$\varphi(v) = \frac{1}{1+e^{-av}}$$

El parámetro *a* permite obtener diferentes funciones con diferentes pendientes. Así, a medida que el parámetro *a* se aproxima a infinito la función se convierte en una simple función threshold. La función sigmoide, además de tener un amplio rango de valores, es diferenciable, cualidad importante en la teoría de redes neuronales.

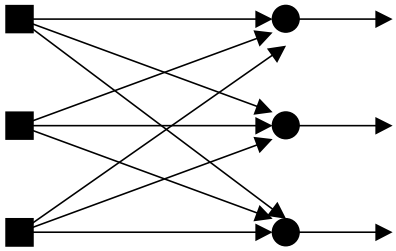
Según la aplicación para la que creemos una red neuronal deberemos escoger la función de activación apropiada.

1.3 Arquitectura de redes

Las neuronas se tienen que unir de cierta forma para componer la red neuronal. La estructura que adopten es también muy importante, ya que está relacionada con el algoritmo de aprendizaje empleado para entrenar la red. En general, podemos distinguir cuatro clases de arquitecturas de redes neuronales:

1.3.1 Redes monocapa

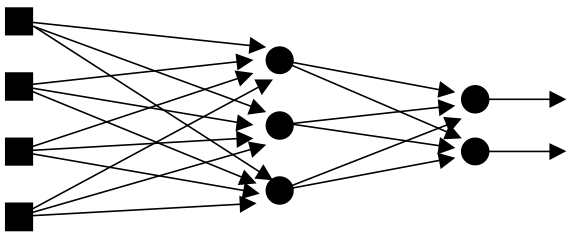
Las redes neuronales están formadas normalmente por varias capas. La estructura más sencilla de una red estructurada en niveles es el caso en el que se tiene un único nivel. En este caso, la red consta de un nivel o capa de entrada de nodos fuente, que actúan como señales de entrada para una capa de salida compuesta de neuronas. Esta arquitectura se denomina de un nivel o capa en referencia a que posee una capa de neuronas. En la siguiente figura se muestra el esquema de una red neuronal de un nivel compuesta por tres nodos de entrada (representados por cuadrados) y tres neuronas (representadas por círculos):



En esta arquitectura de red, los nodos de entrada están conectados con las neuronas de forma que actúan de entrada para éstos, pero no existe conexión en dirección contraria, de ahí que a este tipo de redes se les denomine de tipo *feedforward*.

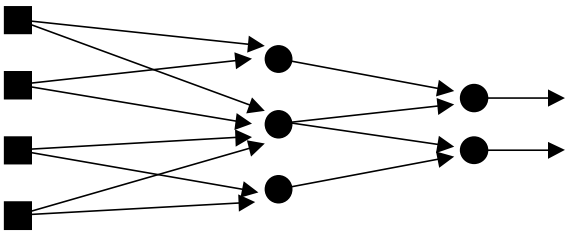
1.3.2 Redes multicapa

Las redes multicapa se distinguen por disponer de varias capas ocultas, es decir, capas intermedias entre los nodos de entrada y los nodos de salida. Los nodos de cómputo presentes en estas capas se denominan así mismo neuronas ocultas. La adición de capas ocultas permite extraer estadísticas de orden alto al disponer de una mayor interconectividad entre las neuronas presentes en la red. En esta estructura, las salidas de una capa de cómputo son empleadas como entradas para la siguiente capa de cómputo. La respuesta global de la red vendrá dada por las señales de salida de las neuronas de la capa de salida. En la siguiente figura se muestra una red de este tipo:



En esta red podemos ver cómo cada nodo de una capa está conectado con cada uno de los nodos de la siguiente capa. En este caso se dice que la red está completamente conectada. En el caso de que no suceda

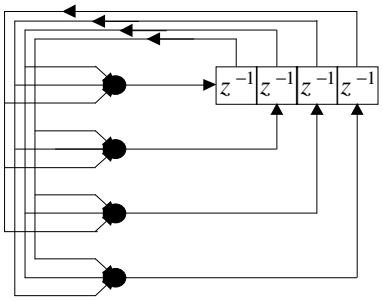
esto, se dice que la red está parcialmente conectada. Una forma particular de red multicapa conectada parcialmente es aquella en las que cada neurona de una capa oculta está conectada a un conjunto de nodos fuente situados en su más inmediata proximidad. En la siguiente figura se muestra un ejemplo de este tipo de red:



Ese conjunto de nodos localizados que alimentan una neurona se dicen que constituyen el campo receptivo de la neurona. Esta red consta de los mismos nodos de entradas, las misma neuronas ocultas y las mismas neuronas en la capa de salida; la única diferencia es la ausencia de ciertos enlaces sinápticos. Tal estructura se dice que está especializa en el sentido de que está orientada hacia un determinado comportamiento.

1.3.3 Redes recurrentes

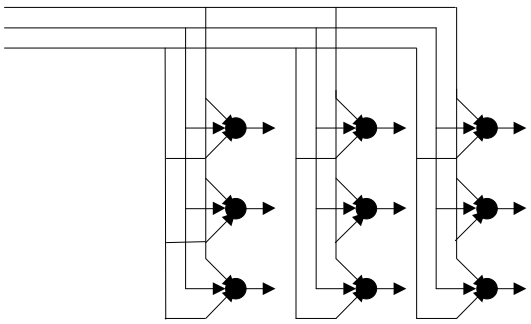
Una red neuronal recurrente se distingue de una red *feedforward* en que al menos posee un bucle de realimentación. Por ejemplo, una red recurrente puede consistir en una única capa de neuronas donde la salida de cada neurona alimenta las entrada del resto de neuronas. En la siguiente figura se muestra un ejemplo de este tipo de redes:



También podemos tener el caso en el que la salida de una neurona alimenta la entrada de esa misma neurona; en este caso diremos que existen lazos de autorrealimentación. La presencia de bucles o lazos de realimentación tiene un gran impacto en la capacidad de aprendizaje

1.3.4 Estructuras Lattice

Una *lattice* consiste en una array de neuronas de una, dos o más dimensiones con un conjunto de nodos de entrada que proporcionan las señales de entrada al array. Una red *lattice* es básicamente una red *feedforward* con las neuronas de salida agrupadas en filas y columnas. En la siguiente figura se muestra un ejemplo de este tipo de estructuras:



1.4 Representación del conocimiento

Cuando en el apartado 2.1 hablábamos del término "conocimiento" en relación con las redes neuronales, ¿qué queríamos decir con ese término?. Podríamos utilizar la siguiente definición: "El conocimiento se refiere a la información almacenada o modelos usados por una persona o máquina para interpretar, predecir o respondes apropiadamente al mundo circundante".

Una tarea importante para una red neuronal es aprender un modelo del entorno en que se encuentre y mantener el modelo suficientemente consistente de forma que se puedan alcanzar los objetivos deseados para la aplicación de interés. El conocimiento del entorno consiste en dos tipos de información:

1. Información a priori. Cuando para una determinada aplicación elegimos una determinada estructura para la red neuronal, estamos incluyendo este tipo de información.

2. Observaciones (medidas). Mediante estas medidas, que serán inherentemente ruidosas, obtendremos un conjunto de ejemplos que usaremos para entrenar la red neuronal.

Cada ejemplo consiste en una señal de entrada y una salida deseada para la red neuronal. Un conjunto de ejemplos de esta clase representarán el conocimiento. Mediante estos ejemplos, el diseño de una red neuronal puede seguir un procedimiento tal que así:

1. Primero, se selecciona una arquitectura apropiada para la red neuronal, con una capa de entrada compuesta por un número de nodos de entrada igual al número de señales de entrada de que dispongamos para nuestra aplicación y una capa de salida con un número de nodos también acordes con nuestra aplicación. Un conjunto de ejemplos se emplean para entrenar la red mediante un algoritmo apropiado. Esta fase se denomina aprendizaje.

2. Segundo, el funcionamiento de la red neuronal se prueba con datos de entrada no utilizados en el entrenamiento y se comprueba el resultado obtenido por la red neuronal con la salida deseada. Esta fase se denomina generalización.

En una red neuronal con una arquitectura determinada, la representación del conocimiento está dada por los valores que toman los parámetros libres de la red, es decir, los valores que toman los pesos sinápticos y los umbrales.

1.4.1 Reglas para la representación del conocimiento

1. *Regla nº 1:*

   Entradas similares pertenecientes a clases similares deben producir representaciones similares dentro de la red y por tanto ser clasificadas como pertenecientes a la misma categoría. La similitud entre las entradas se suele medir mediante un método basado en el concepto de distancia euclídea. Para dos vectores compuestos por elementos reales, se define la distancia euclídea de la siguiente manera:

$$d_{ij} = \|x_i - x_j\| = \left[\sum_{n=1}^{N} (x_{in} - x_{jn})^2 \right]^{1/2}$$

   donde x_{in} y x_{jn} son los n-ésimos elementos de los vectores x_i y x_j.

   Mientras más cercanos sean los distintos elementos de los vectores, menor será la distancia euclídea y mayor será por tanto la similitud entre los dos vectores.

   Otra medida de la similitud entre dos vectores que representen entradas a una red neuronal se basa en el concepto de producto interno. Dados un par de vectores x_i y x_j, se define el producto interno de la siguiente manera:

$$x_i^T x_j = \sum_{n=1}^{N} x_{in} x_{jn}$$

Mientras que la distancia euclídea nos da la longitud de la línea que une los extremos de los vectores, el producto interno nos da la proyección de un vector sobre el otro.

2. *Regla nº 2:*

   Elementos que pertenezcan a clases totalmente diferentes deben dar representaciones diferentes en la red.

3. *Regla nº 3:*

   Si un característica en particular es muy importante, entonces deben existir un gran número de neuronas involucradas en su representación dentro de la red.

4. *Regla nº 4:*

   Información a priori e invariantes deben ser construidas en el propio diseño de la red neuronal, de forma que se simplifique la red no teniendo que aprenderlos. Esto es lo que se llama la especialización de una red neuronal, el que la red esté diseñada estructuralmente pensando en una determina aplicación que posee ciertas características bien conocidas.

1.4.2 Cómo incluir información a priori en el diseño de una red neuronal

No existen reglas bien definidas para desarrollar una estructura especializada incluyendo información a priori en su diseño, pero se puede incluir información a priori mediante dos técnicas:

1. Restringiendo la arquitectura de la red mediante el uso de conexiones locales.
2. Restringir los valores de los pesos sinápticos mediante el uso de compartición de pesos.

La manera en la que se apliquen estas dos técnicas vendrá fuertemente influenciada por la aplicación de interés.

1.4.3 Cómo incluir invarianzas en el diseño de una red neuronal

1. Invarianza por estructura. La invarianza puede ser impuesta en una red neuronal estructurando su diseño apropiadamente. Si nos fijamos en el problema de la clasificación de una imagen, nos interesa que la salida de la red sea invariante ante rotaciones. Esto se forzará creando determinadas conexiones sinápticas entre las neuronas de la red. Si

consideramos w_{ji} el peso sináptico de la neurona j conectada al pixel i de una imagen de entrada, si imponemos que $w_{ji} = w_{jk}$ para todos los pixeles i y k que se encuentren a la misma distancia del centro de la imagen, entonces la red neuronal será invariante ante rotaciones en el plano.

2. Invarianza por entrenamiento. Una red neuronal tiene una habilidad natural para la clasificación de patrones. Esta habilidad se puede explotar para obtener invarianza antes transformaciones de la siguiente manera. La red es entrenada mediante la presentación de un número de diferentes ejemplos del mismo objeto, correspondiéndose estos ejemplos a distintas transformaciones del mismo objeto. Considerando un número suficientemente elevado de muestras y si la red es entrenada para aprender a discriminar las distintas transformaciones del objeto, podemos esperar que la red generalice correctamente otras transformaciones que las presentadas a ella. Sin embargo, desde una perspectiva ingenieril, la invarianza por entrenamiento tiene dos inconvenientes. Primero, cuando una red neuronal ha sido entrenada para reconocer un objeto de manera invariante frente a distintas transformaciones, no es obvio que este entrenamiento permita también a la red reconocer otros objetos pertenecientes a otras clases de forma invariante. Segundo, los requerimientos de cálculo impuestos pueden ser demasiado elevados.

3. Invarianza por extracción de características. Este método consiste en extraer características o propiedades que caractericen la información esencial contenida en un conjunto de datos de entrada, las cuales son invariantes ante transformaciones de la entrada. Esta es la técnica más empleada para clasificadores neuronales.

1.5 Inteligencia artificial y redes neuronales

El objetivo de la inteligencia artificial (IA) es el desarrollo de los algoritmos que requieren las máquinas para realizar tareas que aparentemente requieren *conocimiento* cuando son realizadas por el ser humano. Ésta no es la única definición aceptada para IA. En la definición hemos hablado de *conocimiento* y no de *inteligencia*, ya que entre las tareas abordadas por IA se encuentran la percepción y el lenguaje, así como la solución de problemas y procesos conscientes así como inconscientes.

Todo sistema de inteligencia artificial debe ser capaz de hacer tres cosas: (1) Almacenar conocimientos. (2) Aplicar los conocimientos almacenados para la resolución de problemas. (3) Adquirir nuevos conocimientos a través de la experiencia.

Un sistema IA tiene tres componentes fundamentales:

1. Representación. La característica más distintiva de los sistemas AI puede ser el uso de un lenguaje de estructuras de símbolos para representar tanto conocimientos generales así como conocimientos específicos sobre la solución a un problema. Los símbolos son normalmente formulados en términos familiares, lo que hace las representaciones simbólicas de los sistemas de IA relativamente fáciles de comprender por los usuarios humanos. Conocimiento, usado por los desarrolladores de sistemas de IA, es otro término para datos. Éste puede ser de tipo declarativo o procesal. En una representación declarativa, el conocimiento está representado por una colección estática de datos, con significado por sí mismos para el ser humano. En la procesal, el conocimiento está incluido en un código ejecutable.

2. Razonamiento. En la forma más básica, razonar es la habilidad de solucionar problemas. Para considerar a un sistema capaz de razonar, debe satisfacer ciertas condiciones:

    - El sistema debe ser capaz de expresarse y solucionar un amplio abanico de problemas y tipos de problemas.

    - El sistema debe ser capaz de hacer explícita cualquier información implícita conocida por él.

    - El sistema debe tener un mecanismo de control que determine qué operaciones aplicar a un problema en particular, cuando ha sido obtenida la solución al problema, o cuando un trabajo posterior sobre el problema debería terminarse.

3. Aprendizaje. En un modelo simple de una máquina con capacidad de aprendizaje, el entorno proporciona información a un elemento de aprendizaje, el cual utiliza esa información para realizar mejoras en la base del conocimiento, y finalmente el elemento de desarrollo usa la base del conocimiento para realizar su tarea. La información suministrada por el entorno a la máquina suele ser imperfecta, de forma que el elemento de aprendizaje no sabe de antemano cómo completar detalles que faltan o ignorar detalles irrelevantes. La máquina opera por tanto bajo hipótesis y recibiendo información del elemento de desarrollo, cerrándose un bucle de realimentación, lo cual permite evaluar las hipótesis y modificarlas si fuese necesario.

Una vez familiarizados con las máquinas de inteligencia artificial, ¿cómo podemos compararlas con las redes neuronales?. Esta comparación se realiza a tres niveles:

1. Nivel de explicación. En un sistema IA, el énfasis se realiza en construir representaciones simbólicas que suelen ser discretas y arbitrarias: propiedades abstractas. IA asume la existencia de representaciones mentales. En cambio, en las redes neuronales, el énfasis se realiza sobre el desarrollo de modelos de procesado distribuido en paralelo.

2. Estilo de procesamiento. Mientras que en IA el procesado es secuencial, el procesado en paralelo es una de las características principales de las redes neuronales.

3. Estructura de las representaciones. Mientras que el los sistemas de IA las representaciones simbólicas poseen una estructura cuasi-lingüística, en una red neuronal las representaciones son distribuidas.

2 PROCESOS DE APRENDIZAJE

Entre las muchas propiedades de las redes neuronales, quizás la de mayor importancia sea la habilidad para aprender del entorno, y mejorar su rendimiento a través del aprendizaje. El aprendizaje en una red neuronal es normalmente llevado a cabo mediante un proceso adaptativo, conocido como regla de aprendizaje o algoritmo, donde los pesos de la red son ajustados incrementalmente de forma que se mejore una predefinida medida del rendimiento a lo largo del tiempo. El proceso de aprendizaje se puede ver como un proceso de optimización, en el que se realiza una búsqueda de una solución en un espacio paramétrico multidimensional, que gradualmente optimiza una función objetivo o criterio especificado de antemano.

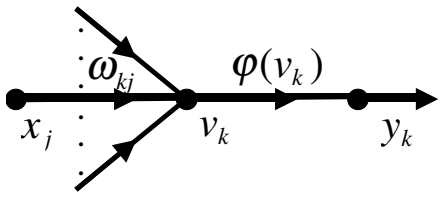
Desde el punto de vista de las redes neuronales, podemos definir el término aprendizaje de la siguiente manera:

" El aprendizaje es el proceso mediante el cual los parámetros libres de una red neuronal son ajustados mediante un proceso continuo de simulación por el entorno en el cual se encuentra la red neuronal. El tipo de aprendizaje viene determinado por la manera en la que se modifican los parámetros de la red".

La definición del proceso de aprendizaje implica los siguientes eventos:

1. La red neuronal es estimulada por el entorno.
2. La red neuronal lleva a cabo cambios como resultado de la estimulación.
3. La red neuronal responde de una manera distinta al entorno, debido a los cambios llevados a cabo en su estructura interna.

Fijémonos en una neurona:



El valor de v_k, conocido como actividad interna de la neurona, depende de la señal de entrada x_j a través del peso sináptico w_{kj}. El valor de este peso sináptico variará en el tiempo de la siguiente manera:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

donde $\Delta w_{kj}(n)$ es el ajuste aplicado al valor del peso sináptico.

Un conjunto de reglas bien definidas para la resolución de un problema de aprendizaje se denomina algoritmo de aprendizaje. Obviamente hay numerosos algoritmos de aprendizaje, cada uno de los cuales tiene sus ventajas e inconvenientes. La principal diferencia entre ellos se basa en la forma de calcular el ajuste Δw_{kj} aplicado a los pesos sinápticos de la red. Otro factor a considerar es la manera en la que la red neuronal se relaciona con su entorno. En este contexto, podemos hablar de un paradigma de aprendizaje refiriéndonos a un modelo del entorno en el cual se encuentra la red neuronal.

2.1 Algoritmos de aprendizaje

2.1.1 Aprendizaje por corrección de error

Denotemos $d_k(n)$ como la respuesta deseada para la neurona k en el instante de tiempo n. Denotemos por $y_k(n)$ la respuesta de esa misma neurona. Esa respuesta es producida por un estímulo x(n) presentado a la entrada de la red donde se encuentra la neurona k. La entrada x(n) y la respuesta deseada $d_k(n)$ para la neurona k constituyen un ejemplo presentado a la red en el instante n. Se asume que este ejemplo y todos los otros ejemplos presentados a la red son generados por un entorno que es probabilístico en naturaleza, aunque la distribución probabilística subyacente sea desconocida.

Normalmente, la respuesta actual $y_k(n)$ de la neurona k es diferente de la respuesta deseada $d_k(n)$. Entonces, podemos definir una señal de error como la diferencia entre la respuesta deseada y la respuesta actual:

$$e_k(n) = d_k(n) - y_k(n)$$

El objetivo del algoritmo de aprendizaje por corrección de error es minimizar una función de coste basada en la señal de error e_k (n), de forma que la respuesta actual de cada neurona de salida en la red se aproxime a la respuesta deseada para dicha neurona en un sentido estadístico. Una vez seleccionada una función de coste, este algoritmo es estrictamente un problema de optimización. Un criterio comúnmente usado para la función de coste es el criterio del error cuadrático medio, definido de la siguiente forma:

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right]$$

donde E es el operador esperanza estadística y la suma es sobre todas las neuronas de salida de la red neuronal. Esta ecuación asume que el proceso es estacionario en sentido amplio. La minimización de la función de coste J respecto a los parámetros de la red lleva al denominado método del descenso del gradiente. Este procedimiento de optimización tiene el problema de que requiere el conocimiento de características estadísticas del proceso. Por ello es que opta por una solución aproximada, en la que se toma el valor instantáneo de la suma de los errores cuadráticos como criterio de interés:

$$\xi(n) = \frac{1}{2}\sum_k e_k^2(n)$$

La red es entonces optimizada minimizando el valor de ξ(n) respecto a los pesos sinápticos. Por tanto, de acuerdo con la regla de aprendizaje por corrección de error, el ajuste realizado al peso sináptico w_{kj} vendrá dado por:

$$\Delta\omega_{kj}(n) = \eta\, e_k(n) x_j(n)$$

donde η es una constante positiva que determina la tasa de aprendizaje. En otras palabras, el ajuste realizado a un peso sináptico es proporcional al producto de la señal de error y la señal de entrada. Se puede ver que este algoritmo se comporta como un bucle de realimentación. Hay que tener cuidado con el valor de la tasa de aprendizaje η que se toma, ya que hay que asegurar la estabilidad del proceso de aprendizaje. Este parámetro tiene un fuerte impacto no solo en la velocidad de convergencia, sino en la propia convergencia. Si se toma un valor elevado, la velocidad de aprendizaje es elevada, pero se corre el riesgo de que el proceso diverja y el sistema se haga inestable.

Una representación de la función de coste J frente a los pesos sinápticos que caracterizan la red neuronal consiste en una superficie multidimensional conocida como superficie de rendimiento del error o simplemente superficie de error. Dependiendo del tipo de unidades de procesado usadas para construir la red neuronal, podemos distinguir dos situaciones diferentes:

1. La red neuronal está compuesta completamente por unidades de procesado lineal, en cuyo caso la superficie de error es exactamente una función cuadrática de los pesos de la red. La superficie de error tendrá un único mínimo.

2. La red neuronal consta de unidades de procesado no lineales, en cuyo caso la superficie de error posee un mínimo global (o varios) así como mínimos locales.

En ambos casos el objetivo del algoritmo es comenzar desde un punto arbitrario de la superficie de error y moverse hasta el mínimo global paso a paso. En el primer caso este objetivo es alcanzable, pero no ocurre lo mismo en el segundo caso, ya que el algoritmo se puede quedar atrapado en un mínimo local y por tanto no poder alcanzar nunca el mínimo global.

2.1.2 Aprendizaje Hebbiano

El postulado sobre el aprendizaje de Hebb se puede reescribir de la siguiente forma:

1. Si dos neuronas a ambos lados de una sinapsis (conexión) son activadas simultáneamente, entonces la fuerza de la sinapsis es incrementada selectivamente.

2. Si dos neuronas a ambos lados de una sinapsis son activadas asíncronamente, entonces la sinapsis es selectivamente debilitada o eliminada.

Una sinapsis con tal comportamiento se denomina sinapsis hebbiana. Más precisamente, se puede definir una sinapsis hebbiana como una sinapsis que usa un mecanismo interactivo fuerte, altamente local y dependiente del tiempo para incrementar la eficiencia de la sinapsis como una función de la correlación entre las actividades presinápticas y postsinápticas. De esta definición podemos deducir las cuatro siguientes propiedades que caracterizan una sinapsis hebbiana:

1. Mecanismo dependiente del tiempo. Este mecanismo se refiere al hecho de que las modificaciones en una sinapsis hebbiana dependen del momento exacto en el que ocurren las actividades presinápticas y postsinápticas.

2. Mecanismo local. Las modificaciones que se realizan en una sinapsis dependen como ya hemos dicho de las actividades presinápticas y postsinápticas.

3. Mecanismo interactivo. De nuevo, como la ocurrencia de una cambio en una sinapsis depende de la interacción de las actividades presinápticas y postsinápticas, no podemos hacer una predicción a partir de uno de los dos niveles de actividad.

4. Mecanismo conjuntivo y correlacionado. A veces se denomina a una sinapsis hebbiana como sinapsis conjuntiva por el hecho de la necesidad de ocurrencia al mismo tiempo de las actividades presinápticas y postsinápticas. También, si pensamos en términos estadísticos, la correlación a lo largo del tiempo de ambas actividades se puede ver como la responsable de un cambio en una sinapsis.

Si consideramos una sinapsis hebbiana como aquella que se refuerza cuando las actividades presinápticas y postsinápticas están correladas positivamente y se debilita cuando están correlacionadas negativamente, se denomina sinapsis anti-hebbiana como aquella que se debilita cuando están correladas positivamente y se refuerza cuando están correladas negativamente.

De acuerdo con el postulado de Hebb, el ajuste aplicado a un peso sináptico w_{kj} en el instante n se puede expresar de la siguiente manera:

$$\Delta\omega_{kj} = F(y_k(n), x_j(n))$$

donde F es una función de las actividades presinápticas y postsinápticas ($x_j(n)$ y $y_k(n)$ respectivamente). Como un caso especial, podemos escribir que:

$$\Delta\omega_{kj}(n) = \eta\, y_k(n) x_j(n)$$

donde η es una constante positiva que determina la tasa de aprendizaje. Esta última ecuación es la forma más simple para el cambio de los pesos sinápticos, expresada como el producto de la señal de entrada y la señal de salida. Esta ecuación enfatiza claramente la naturaleza correlativa de las sinapsis hebbianas.

Si nos detenemos a analizar la última ecuación presentada podemos comprobar que la repetida aplicación de una señal de entrada lleva a un crecimiento exponencial que conduce al peso sináptico a la saturación. Para evitar esta situación, hay que imponer un límite al crecimiento del peso sináptico. Un método para esto es introducir un factor de olvido no lineal en la fórmula de ajuste de los pesos sinápticos, quedando ésta como sigue:

$$\Delta\omega_{kj}(n) = \eta\, y_k(n) x_j(n) - \alpha\, y_k(n)\omega_{kj}(n)$$

donde α es una constante positiva. Esta ecuación se puede reescribir de la siguiente manera:

$$\Delta\omega_{kj}(n) = \alpha\, y_k(n)\left[cx_j(n) - \omega_{kj}(n)\right]$$

A partir de esta ecuación podemos ver que en función del valor de $x_j(n)$, el peso sináptico se podrá ver tanto incrementado como decrementado, dándose el punto de equilibrio cuando $x_j(n) = w_{kj}(n) / c$. Además, con esta aproximación se ha evitado la inestabilidad en el valor del peso sináptico.

Otra manera de formular el postulado de Hebb es en términos estadísticos, considerando que los cambios en los pesos sinápticos son proporcionales a la covarianza entre las actividades presinápticas y postsinápticas, de forma que:

$$\Delta\omega_{kj}(n) = \eta\,\mathrm{cov}\left[y_k(n), x_j(n)\right] = \eta E\left[\left(y_k(n) - \bar{y}_k\right)\left(x_j(n) - \bar{x}_j\right)\right]$$

Las variables x_j e y_j son los valores medios de las actividades presinápticas y postsinápticas, y permiten que los pesos sinápticos puedan tanto incrementarse como decrementarse.

2.1.3 Aprendizaje competitivo

En el aprendizaje competitivo, las neuronas de salida de la red neuronal compiten entre sí para ser la única activa. Mientras que en una red neuronal basada en aprendizaje hebbiano varias neuronas pueden estar activadas simultáneamente, en el caso de aprendizaje competitivo solo una neurona puede estar activa en un instante de tiempo. Es esta característica la que hace que este método sea adecuado para la clasificación de patrones.

En un algoritmo de aprendizaje competitivo hay tres elementos básicos:

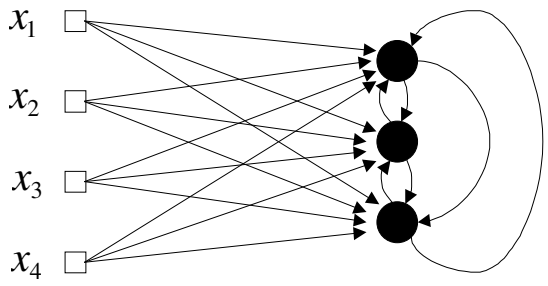
1. Un conjunto de neuronas que son exactamente iguales excepto por algunos pesos sinápticos distribuidos aleatoriamente, y que por tanto responderán de manera diferente ante un conjunto de patrones de entrada.

2. Un límite impuesto en la fuerza de cada neurona.

3. Un mecanismo que permita a las neuronas competir por el derecho a responder a un subconjunto de entradas dado, de modo que solo una neurona, o solo una neurona por grupo, está activa al mismo tiempo.

Las neuronas individuales de la red aprenden a especializarse en conjuntos de patrones similares, y por tanto se convierten en detectores de características.

En la forma más simple de aprendizaje competitivo, la red neuronal tiene una sola capa de neuronas de salida, cada una de las cuales está conectada a todos los nodos de entrada. La red puede incluir conexiones laterales entre las neuronas, como se puede ver en la siguiente figura:



En esta estructura descrita, las conexiones laterales realizan una inhibición lateral, con cada neurona intentando inhibir la neurona con la que está conectada lateralmente. El resto de conexiones son excitantes.

Para la neurona ganadora, su nivel de actividad interno v_j para un patrón de entrada x determinado debe ser el mayor entre todas las neuronas en la red. La señal de salida de la neurona ganadora se pone a uno; las señales de salida del resto de neuronas perdedoras se ponen a cero.

Sea w_{ji} es el peso sináptico que conecta el nodo de entrada i a la neurona j. Cada neurona tiene asignado una cantidad fija de peso sináptico (todos los pesos sinápticos son positivos), distribuido entre sus nodos de entrada, de forma que:

$$\sum_i \omega_{ji} = 1 \quad \forall j$$

Una neurona aprende pasando pesos sinápticos desde los nodos de entrada inactivos a los activos. Si una neurona no responde a un patrón de entrada en particular, no se lleva a cabo aprendizaje en esa neurona. Si una neurona gana la competición, cada nodo de entrada de esa neurona renuncia a parte de su peso sináptico, y ese peso es distribuido equitativamente entre los nodos de entrada activos. De acuerdo con la regla de aprendizaje competitivo estándar, el cambio aplicado al peso sináptico w_{ji} viene dado por:

$$\Delta\omega_{ji} = \begin{cases} \eta\left(x_i - w_{ji}\right) & si la neurona\ j\ gana la competición \\ 0 & si la neurona\ j\ pierde la competición \end{cases}$$

Esta regla tiene el efecto global de mover el vector de peso sináptico w_j de la neurona ganadora j hacia el patrón de entrada x.

2.1.4 Aprendizaje de Boltzmann

En una máquina de Boltzmann, las neuronas constituyen una estructura recurrente, y operan de una manera binaria, de forma que las neuronas pueden estar en un estado activo, designado por un +1, o en un estado inactivo, designado por un –1. La máquina está caracterizada por una función de energía E, cuyo valor está determinado por los estados particulares ocupados por las neuronas individuales de la máquina, como se puede ver en la siguiente expresión:

$$E = -\frac{1}{2}\sum_{i_{i\neq j}}\sum_{j}\omega_{ji}s_j s_i$$

donde s_i es el estado de la neurona i, y w_{ji} es el peso sináptico que conecta la neurona i a la neurona j. El hecho de que i ≠ j significa que ninguna neurona posee autorrealimentación. La máquina funciona escogiendo una neurona al azar, la neurona j por ejemplo, en algún paso del proceso de aprendizaje, y cambiando el estado de esta neurona del estado s_j al estado $-s_j$ a cierta temperatura con probabilidad:

$$W(s_j \rightarrow -s_j) = \frac{1}{1+\exp(-\Delta E_j / T)}$$

donde ΔEj es el cambio de energía resultado de dicho cambio. T no es una temperatura física, sino una pseudotemperatura. Si esta regla es aplicada repetidamente, la máquina alcanzará equilibrio térmico.

Las neuronas de una máquina de Boltzmann se dividen en dos grupos: visibles y ocultas. Las neuronas visibles proporcionan un interfaz entre la red y el entorno en que opera ésta, mientras que las neuronas ocultas operan libremente. Hay que considerar dos modos de operación :

1. Sujeto a condiciones, en el que las neuronas visibles están sujetas a estados específicos determinados por el entorno.

2. Libre de condiciones, en el que todas las neuronas operan libremente.

Denotemos por ρ^{+}_{ji} la correlación entre los estados de las neuronas i y j, condicionado porque la red se encuentre sujeta a condiciones. Denotemos por ρ^{-}_{ji} la correlación entre los estados de las neuronas i y j cuando la red opera libre de condiciones. Ambas correlaciones son promediadas entre todos los estados posibles de la máquina cuando se encuentra en equilibrio térmico. Estas correlaciones se definen de la siguiente manera :

$$\rho_{ji}^{+} = \sum_{\alpha}\sum_{\beta} P_{\alpha\beta}^{+} s_{j|\alpha\beta} s_{i|\alpha\beta}$$

$$\rho_{ji}^{-} = \sum_{\alpha}\sum_{\beta} P_{\alpha\beta}^{-} s_{j|\alpha\beta} s_{i|\alpha\beta}$$

donde $s_i|_{\alpha\beta}$ representa el estado de la neurona i cuando las neuronas visibles de la máquina están en el estado α y las neuronas ocultas se encuentran en el estado β. El factor $P^{+}_{\alpha\beta}$ es la probabilidad condicional de que las neuronas visibles estén en el estado α y las neuronas ocultas se encuentren a la vez en el estado β cuando la máquina se encuentra sujeta a condiciones; y $P^{-}_{\alpha\beta}$ es la probabilidad condicional de que las neuronas visibles estén en el estado α y las neuronas ocultas se encuentren a la vez en el estado β cuando la máquina se encuentra libre de condiciones. Entonces, de acuerdo con la regla de aprendizaje de Boltzmann, el cambio aplicado al peso sináptico w_{ji} de la neurona i a la neurona j está dado por:

$$\Delta\omega_{ji} = \eta(\rho_{ji}^{+} - \rho_{ji}^{-}) \quad i \neq j$$

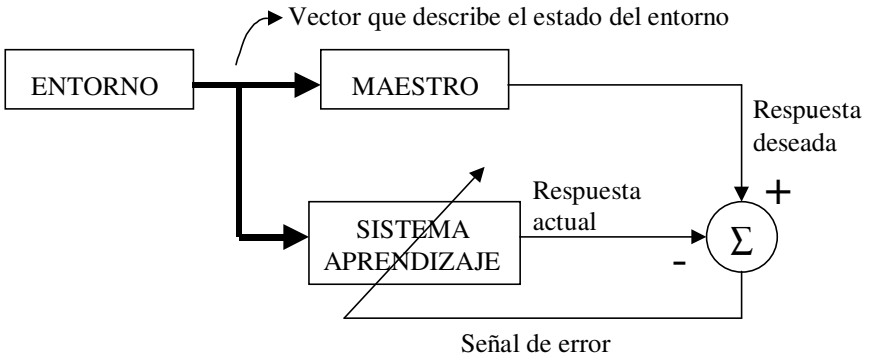
donde η es la tasa de aprendizaje. Hacer notar que ρ^{+}_{ji} y ρ^{-}_{ji} pueden tomar los valores –1 y +1.

2.2 Paradigmas de aprendizaje

Hablamos de paradigma del aprendizaje refiriéndonos al modelo del entorno en el cual opera una red neuronal. A continuación veremos las tres clases más básicas de paradigmas del aprendizaje.

2.2.1 Aprendizaje supervisado

Un ingrediente esencial del aprendizaje supervisado o activo es la disponibilidad de un profesor externo, como se puede apreciar en la siguiente figura:



Conceptualmente, podemos pensar del profesor como conocedor del entorno que es representado por un conjunto de ejemplos entrada-salida. El entorno es, sin embargo, desconocido para la red neuronal. Supongamos ahora que el profesor y la red neuronal son sometidos a un vector de entrenamiento sacado del entorno. El profesor puede proporcionarle a la red la respuesta deseada para ese vector de entrenamiento. Los parámetros de la red son ajustados bajo la influencia conjunta del vector de entrenamiento y la señal de error; la señal de error es definida como la diferencia entre la respuesta actual de la red y la respuesta deseada. Este ajuste es llevado a cabo de una manera iterativa con el objetivo de emular el profesor. En otras palabras, el conocimiento del entorno por parte del profesor es transferido a la red neuronal en la mayor medida posible. Cuando esta condición es alcanzada, podemos prescindir del profesor y dejar a la red tratar con el entorno por sí misma.

La forma de aprendizaje supervisado que acabamos de describir es en realidad el proceso de aprendizaje por corrección del error. Es un sistema realimentado, pero el entorno desconocido no se encuentra en el bucle. Se puede tomar como medida del rendimiento del sistema el error cuadrático medio, definido como función de los parámetros libres de la red. Como ya vimos, esta función puede ser visualizada como una superficie de error. Con la ayuda del profesor, la red deberá ir moviendo el punto de operación hacia el mínimo global. Un sistema de aprendizaje supervisado es capaz de realizar esto gracias a la información que posee acerca del gradiente de la superficie de error correspondiente al actual comportamiento del sistema. El gradiente de la superficie de error en cualquier punto es un vector que apunta hacia la dirección de mayor descenso. De hecho, en el caso de aprendizaje supervisado a partir de ejemplos, el sistema utiliza una estimación instantánea del vector gradiente, de forma que dado un conjunto adecuado de ejemplos entrada-salida y suficiente tiempo para el aprendizaje, un sistema de aprendizaje supervisado es normalmente capaz de realizar tareas como la clasificación de patrones y aproximación de funciones satisfactoriamente.

Ejemplos de algoritmos de aprendizaje supervisado son el LMS y su generalización conocida como el algoritmo de propagación hacia atrás (Backpropagation ó BP). El aprendizaje supervisado se puede llevar a cabo de dos maneras:

1. Off-line: en este caso se utiliza una facilidad de cómputo externa para diseñar el sistema de aprendizaje supervisado. Una vez conseguido el rendimiento deseado, el diseño del sistema se congela, de manea que la red funciona de manera estática.

2. On-line: en este caso el procedimiento de aprendizaje es implementado dentro del sistema, de forma que el aprendizaje se lleva a cabo en tiempo real, por lo que la red neuronal es dinámica.

Una desventaja del aprendizaje supervisado es el hecho de que sin el profesor la red neuronal no puede aprender nuevas estrategias para situaciones particulares que no están cubiertas por el conjunto de ejemplos empleados para el entrenamiento.

2.2.2 Aprendizaje reforzado

El aprendizaje reforzado es un aprendizaje en tiempo real de un mapeo de parejas entrada-salida a través de un proceso de prueba y error diseñado para maximizar un índice de rendimiento escalar denominado señal de refuerzo.

La idea de aprendizaje reforzado tiene sus orígenes en estudios experimentales sobre el aprendizaje de animales en psicología. Veamos la ley del efecto de Thorndike:

"*De diversas respuestas ante la misma situación, aquellas que son acompañadas o seguidas inmediatamente de satisfacción para el animal, estarán más conectadas con esta situación, de forma, que si ocurre la misma situación, serán más propensas a darse; aquellas que estén acompañadas o inmediatamente seguidas de algo desagradable para el animal, verán debilitadas su conexión con dicha situación, de forma que si ocurre la misma situación, serán menos propensas a ocurrir*."

Aunque no se puede decir que este principio proporcione un modelo completo del comportamiento biológico, su simplicidad y proximidad al sentido común lo han hecho una regla de aprendizaje muy influyente.

El paradigma del aprendizaje reforzado puede ser de dos tipos:

1. No asociativo: el sistema de aprendizaje tiene la tarea de seleccionar una única acción óptima más que asociar diferentes acciones con diferentes estímulos. En un problema de aprendizaje de este tipo, el reforzamiento es la única entrada que recibe el sistema del entorno.

2. Asociativo: el entorno proporciona más información aparte del reforzamiento. Además, se debe aprender un mapeado en la forma de asociación de estímulos y acciones.

2.2.3 Aprendizaje no supervisado o auto-organizado

En el aprendizaje no supervisado no hay profesor externo para controlar el proceso de aprendizaje. No hay ejemplos específicos de la función a ser aprendida por la red. En su lugar, se tiene una medida de la calidad de la representación que la red debe aprender independiente de la tarea, y los parámetros libres de la red son optimizados según esa medida. Una vez que la red se ha ajustado a las regularidades

estadísticas de los datos de entrada, es capaz de formar representaciones internas para las características de la entrada.

Para realizar un aprendizaje no supervisado, se debe usar una regla de aprendizaje competitivo. Por ejemplo, podemos usar una red neuronal que consiste en dos capas, una capa de entrada y una capa competitiva. La capa de entrada recibe los datos disponibles. La capa competitiva consiste en neuronas que compiten unas contra otras por la oportunidad de responder a características contenidas en los datos de entrada.

2.2.4 Comparativa entre aprendizaje supervisado, reforzado y no supervisado

En un sistema de aprendizaje supervisado, el profesor proporciona información directa sobre cómo el sistema debería cambiar el comportamiento para mejorar el rendimiento. Esta información es de naturaleza local, definida por una estimación instantánea del gradiente de la superficie de error en el punto de operación actual, lo cual permite saber cómo modificar los valores de los parámetros de la red neuronal para incrementar el rendimiento.

En un problema de aprendizaje reforzado no hay profesor que proporcione información del gradiente durante el aprendizaje. La única información disponible es el refuerzo aportado por el entorno. El sistema tiene que hacer cosas y ver qué pasa para obtener información acerca del gradiente. El refuerzo es un escalar, y aunque sirve para evaluar el comportamiento, no indica si se puede mejorar el rendimiento o cómo el sistema debería cambiar su comportamiento. El sistema busca información acerca de la dirección en la que debe modificar su comportamiento en las propiedades intrínsecas del entorno mediante un proceso de prueba y error. Esto hace que un sistema de este tipo se comporte de una manera más lenta.

Uno de los algoritmos más usados para realizar aprendizaje supervisado es el de Backpropagation, en el cual se pueden distinguir dos fases: en la primera, las señales de entrada se propagan por la red capa a capa, produciendo cierta respuesta a la salida de la red; en la segunda, la señal de salida se compara con la salida deseada generando una señal de error, que es propagada hacia atrás por la red, ajustándose los parámetros de la red para minimizar la suma de los errores cuadráticos. Cuando se tienen varias capas, el cómputo se puede hacer demasiado lento debido a la gran interactividad entre los pesos sinápticos.

Una posible solución a este problema es utilizar aprendizaje no supervisado, aprovechando la habilidad de este tipo de redes de formar representaciones internas que modelen la estructura subyacente de los datos de entrada, de forma que las respuestas correctas puedan ser asociadas con las representaciones internas de la red de una manera más rápida. El uso conjunto de procedimientos de aprendizaje supervisado y no supervisado pueden proporcionar una solución más aceptable que el aprendizaje supervisado por sí solo, sobre todo si el tamaño del problema es considerable.

2.3 Tareas de aprendizaje

La elección de un proceso de aprendizaje en particular está muy influenciado por la tarea de aprendizaje para la que se requiere la red neuronal. Veamos diferentes tareas de aprendizaje que se pueden llevar a cabo:

1. Aproximación. Supongamos que tenemos una cierta relación no lineal entre una entrada y un salida, dada por la siguiente expresión:

$$d = g(x)$$

donde x es el vector de entrada y d es un escalar de salida. Se supone que la función g (·) es desconocida. El objetivo es diseñar una red neuronal que aproxime esa función no lineal dado un conjunto de ejemplos compuestos por pares entrada-salida. Este problema es un perfecto candidato para el aprendizaje supervisado.

2. Asociación. Esta tarea puede tener dos formas, denominadas autoasociativa y heteroasociativa. En la autoasociación, la red neuronal debe almacenar unos patrones por la presentación repetida de estos a la red. Más tarde, se le presentarán a la red versiones de estos patrones distorsionadas por ruido o descripciones parciales de estos, y la tarea de la red será obtener el patrón original. La heteroasociación se diferencia de la autoasociación en que un conjunto arbitrario de patrones de entrada es emparejado con otro conjunto arbitrario de patrones de salida. Mientras que la autoasociación implica el uso de aprendizaje no supervisado, el tipo de aprendizaje implicado en la heteroasociación es supervisado.

3. Clasificación de patrones. En esta tarea existe un conjunto de clases en que deben ser clasificadas las entradas. Para resolverlo, la red neuronal tiene una primera fase de entrenamiento, durante la que se le presenta un conjunto de patrones de entrada junto con la categoría a la que pertenecen. Más tarde, se le presenta a la red un patrón de entrada nunca visto, y la tarea de la red neuronal es clasificar este patrón como perteneciente a una de las clases. Tal como lo hemos descrito, esta tarea se corresponde con un problema de aprendizaje supervisado. La ventaja de usar una red neuronal para la clasificación de patrones es que se pueden construir límites de decisión no lineales, ofreciendo un método práctico para resolver problemas de clasificación de patrones de alta complejidad.

4. Predicción. Esta es una de las tareas más básicas en las que a partir de ciertas muestras pasadas se trata de predecir la siguiente. La predicción se puede realizar usando aprendizaje por corrección de error de manera no supervisada en el sentido de que los ejemplos de

entrenamiento están sacados directamente de la serie temporal de muestras. El error cometido en la predicción se podrá calcular de la siguiente manera:

$$e(n) = x(n) - \hat{x}(n \mid n-1,...,n-M)$$

La predicción puede ser vista como una forma de construcción de un modelo en el sentido de que cuanto menor se haga el error de predicción, mejor servirá la red neuronal como modelo físico del proceso estocástico subyacente responsable de la generación de la serie temporal. Cuando este proceso es de naturaleza no lineal, las redes neuronales se convierten en un método muy potente para la resolución de problemas de predicción.

5. Control. El control de un proceso es otra tarea que encaja con el uso de redes neuronales. El cerebro humano es una prueba viviente de que es posible construir un controlador generalizado que se aproveche de la ventaja del hardware distribuido en paralelo, que es capaz de manejar muchos miles de actuadores en paralelo (fibras de los músculos) y que puede manejar no linealidades y ruido.

6. Beamforming. Es una forma de filtrado espacial cuyo objetivo es localizar una señal objetivo dentro de un fondo o interferencia aditiva. En radares, esta tarea se complica por dos motivos:

    - La señal objetivo procede de una dirección desconocida.
    - No hay información disponible a priori de las características estadísticas de la interferencia.

    Para solventar estos problemas, se hace necesario el uso de un array de antenas diseñados para dirigir el lóbulo principal hacia el objetivo y colocar nulos sobre las direcciones desconocidas de las señales de interferencia para cancelar estas.

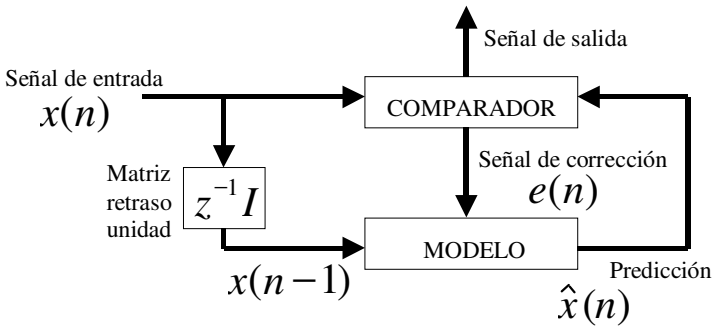
2.4 Adaptación y aprendizaje

El espacio es una dimensión fundamental del proceso de aprendizaje; el tiempo es la otra. La naturaleza espacio-temporal del aprendizaje es ejemplificada por muchas de las tareas de aprendizaje. Cuando una red neuronal opera en un entorno estacionario, las características estadísticas esenciales del entorno pueden ser aprendidas en teoría por la red bajo la supervisión de un profesor. Los pesos sinápticos de la red pueden ser calculados sometiendo a la red a un proceso de entrenamiento con un conjunto de datos representativos del entorno. Una vez terminado este proceso, los pesos sinápticos deberían capturar la

estructura estadística subyacente en el entorno. Por tanto, el sistema de aprendizaje se basa en la memoria para recordar y utilizar experiencias pasadas.

Sin embargo, frecuentemente, el entorno de interés es de naturaleza no estacionaria, lo que significa que los parámetros estadísticos de las señales que transportan la información generada por el entorno varían con el tiempo. En tal situación, los métodos tradicionales de aprendizaje supervisado se muestran inadecuados, ya que la red no dispone de suficientes medios para seguir las variaciones estadísticas del entorno en que opera. Para solventar este problema, la red debe ser capaz continuamente de adaptar sus parámetros libres ante variaciones en las señales de entrada en tiempo real. Así, un sistema adaptativo responde a cada entrada distinta como uno nuevo. Dada esa capacidad, se puede decir que cuando un sistema se encuentra operando en un entorno no estacionario, cuanto más adaptativo lo hagamos, siempre de una manera controlada, más probable será que opere mejor.

Entonces, ¿cómo puede una red neuronal adaptar su comportamiento a la estructura temporal de las señales de entrada en su espacio de comportamiento?. En la siguiente figura se muestra una disposición conceptual de un único nivel de procesado neuronal que puede ser usado para proporcionar esta adaptación:



El elemento denominado MODELO funciona como un predictor en el sentido de que usa la experiencia obtenida a lo largo del tiempo para calcular qué es lo que va a ocurrir, es decir, a partir de los valores pasados de la señal de entrada, reflejados en el vector x (n-1) y en los parámetros de la red en el instante n-1, el modelo proporciona una estimación del actual vector de entrada. El comparador calculará la diferencia entre la predicción y el actual vector de entrada; la diferencia se denominará innovación y representa la información nueva contenida en la señal de entrada en el instante de tiempo del cómputo. Si la innovación vale cero, no se produce nueva información en el sentido de que el modelo sabía qué era lo que iba a suceder. Si la innovación es distinta de cero, significa que ha ocurrido algo inesperado, y el sistema debería tratar de seguirlo.

La innovación es explotada de dos formas:

1. La innovación proporciona una señal de corrección que al ser aplicada al modelo provoca que este ajuste sus parámetros libres y por tanto aprenda qué está ocurriendo en el entorno que lo rodea.

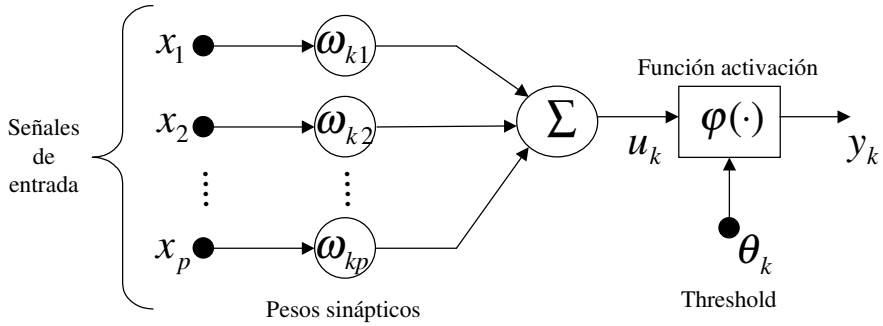
2. La innovación está disponible como salida para ser transferida al siguiente nivel de procesado neuronal para su interpretación. Repitiendo esta operación nivel a nivel, la información procesada tiende a ser progresivamente de mayor calidad, ya que cada nivel neuronal procesa solo la información que no pudo ser procesada en los niveles inferiores.

Podemos por tanto incluir estructura temporal en el diseño de una red neuronal manteniendo la red neuronal en una entrenamiento continuo con ejemplos ordenados temporalmente. De acuerdo con esta aproximación, una red neuronal puede ser vista como un filtro adaptativo no lineal que representa una generalización de filtros adaptativos lineales. Alternativamente, la estructura temporal puede ser aprendida extendiendo métodos tradicionales de aprendizaje supervisado.

3 EL PERCEPTRÓN

El perceptrón es la forma más simple de una red neuronal usada para la clasificación de un tipo especial de patrones denominados linealmente separables (patrones que se encuentran en los lados opuestos de un hiperplano). Rosenblatt probó que si los patrones (vectores) usados para entrenar el perceptrón están sacados de dos clases linealmente separables, entonces el algoritmo del perceptrón converge y coloca la superficie de decisión en la forma de un hiperplano entre las dos clases. La prueba de la convergencia del algoritmo es conocida como el teorema de la convergencia del perceptrón.

Básicamente, consiste de una única neurona con pesos sinápticos y umbral ajustables, como se muestra en la siguiente figura:



Este perceptrón está limitado a realizar la clasificación de patrones en solo dos clases. Expandiendo la capa de salida para incluir más de una neurona, podemos realizar la clasificación de patrones en más

clases. Sin embargo, las clases deben ser linealmente separables para que al perceptrón funcione correctamente.

3.1 Consideraciones básicas

Como ya hemos visto con anterioridad, el modelo de una neurona consiste de un combinador lineal seguido por un limitador. El nodo encargado de hacer la suma realiza una combinación lineal de las entradas aplicadas a sus sinapsis y además aplica el umbral externo. La suma resultante es aplicada al limitador. La neurona produce una salida de +1 si la entrada del limitador es positiva y –1 si es negativa.

En la figura que vimos con anterioridad del modelo de una neurona, los pesos sinápticos del perceptrón son denotados por w_1, w_2, ..., w_p. Correspondientemente, las entradas aplicadas al perceptrón son denotadas por x_1, x_2, ..., x_p. El umbral aplicado externamente es denotado por θ. Por lo tanto, podemos ver que la salida del combinador lineal viene dada por:

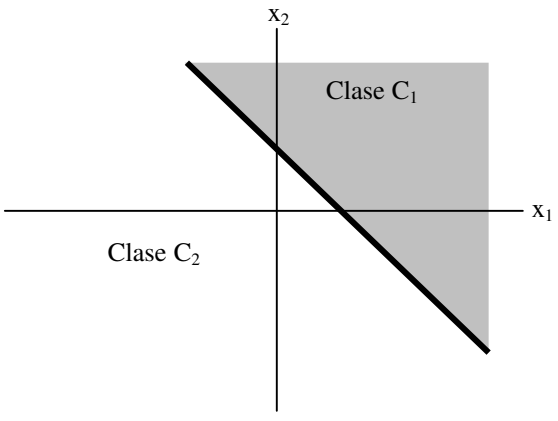
$$v = \sum_{i=1}^{p} w_i x_i - \theta$$

El propósito del perceptrón es clasificar el conjunto de estímulos aplicados externamente x_1, x_2, ..., x_p en una de dos clases, C_1 ó C_2. La regla de decisión para la clasificación es asignar el punto representado por las entradas x_1, x_2, ..., x_p a la clase C_1 si la salida del perceptrón es +1 y a la clase C_2 si es –1.

Para introducirnos algo más en el comportamiento de un clasificador de patrones, es costumbre representar un mapa de regiones de decisión en el espacio de señal de dimensión p. En el caso de un perceptrón elemental, hay dos regiones de decisión separadas por un hiperplano definido por:

$$\sum_{i=1}^{p} w_i x_i - \theta = 0$$

En la siguiente figura se ilustra el caso de dos variables de entrada:



donde el límite de decisión viene dado por:

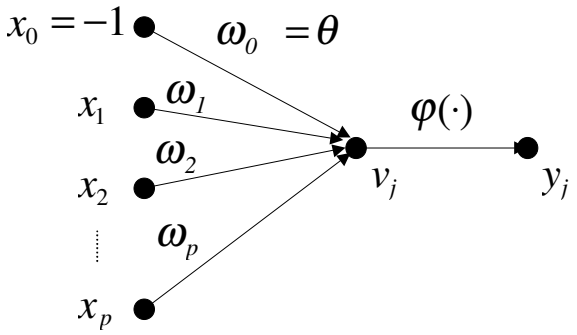
$$w_1 x_1 + w_2 x_2 - \theta = 0$$

Cualquier punto (x_1,x_2) que caiga sobre la línea de decisión es asignado a la clase C_1, mientras que cualquier punto (x_1,x_2) que caiga por debajo de la línea de decisión será asignado a la clase C_2. Notar también que el umbral θ simplemente realiza un desplazamiento de la línea de decisión lejos del origen.

Los pesos sinápticos w_1, w_2, ..., w_p del perceptrón pueden ser fijos o adaptarse de manera iterativa. Para la adaptación, se puede usar una regla de corrección de error conocida como el algoritmo de convergencia del perceptrón, el cual describimos a continuación.

3.2 Teorema de la convergencia del perceptrón

Para el desarrollo del algoritmo de aprendizaje por corrección del error para un perceptrón de una sola capa, es más conveniente trabajar con un modelo de grafo de señal modificado como el de la siguiente figura:



En este modelo, el umbral es tratado como otro peso sináptico con entrada fija a −1. Entonces, podemos definir el vector de entrada como:

$$x(n) = \left[-1, x_1(n), x_2(n), ..., x_p(n)\right]^T$$

De la misma manera, podemos definir el vector de pesos sinápticos de la siguiente manera:

$$w(n) = \left[\theta(n), w_1(n), w_2(n), ..., w_p(n)\right]^T$$

Así, la salida del combinador lineal se puede escribir de una manera muy compacta:

$$v(n) = w^T(n) \cdot x(n)$$

Para un n fijado, la ecuación $w^T x = 0$, dibujada en un espacio de dimensión p con coordenadas x_1, x_2, ..., x_p, define un hiperplano como la superficie de decisión entre dos clases diferentes de entradas.

Supongamos que las variables de entrada de un perceptrón de una capa provienen de dos clases linealmente separables que caen en los lados opuestos de un hiperplano. Denotemos por X_1 al subconjunto de vectores de entrenamiento $x_1(1)$, $x_1(2)$,... que pertenecen a la clase C_1, y denotemos por X_2 al subconjunto de vectores de entrenamiento $x_2(1)$, $x_2(2)$,... que pertenecen a la clase C_2. La unión de X_1 y X_2 es el conjunto completo de entrenamiento X. Dados los conjuntos de vectores X_1 y X_2 para entrenar el clasificador, el proceso de entrenamiento incluye el ajuste del vector de pesos w de tal forma que las dos clases C_1 y C_2 sean separables. Estas dos clases se dicen que son separables linealmente si existe un vector de pesos sinápticos w realizable. A la inversa, si se sabe que las dos clases C_1 y C_2 son separables, entonces existe un vector de pesos sinápticos w de forma que podamos establecer que:

$w^T \cdot x \geq 0$ para cualquier vector de entrada x que pertenezca a la clase C_1

y

$w^T \cdot x < 0$ para cualquier vector de entrada x que pertenezca a la clase C_2

Dados los subconjuntos de vectores de entrenamiento X_1 y X_2, el problema del entrenamiento del perceptrón elemental es entonces encontrar un vector de pesos w de forma que se satisfagan las dos inecuaciones anteriores.

El algoritmo para adaptar el vector de pesos del perceptrón elemental puede ser formulado ahora de la siguiente manera:

1. Si el n-ésimo miembro del vector de entrenamiento, x(n), es correctamente clasificado por el vector de pesos w(n) calculado en la n-ésima iteración del algoritmo, no se lleva a cabo ninguna corrección al vector de pesos del perceptrón, como se muestra a continuación:

   $w(n+1) = w(n)$ si $w^T(n) \cdot x(n) \geq 0$ y $x(n)$ pertenece a la clase C_1

   y

   $w(n+1) = w(n)$ si $w^T(n) \cdot x(n) < 0$ y $x(n)$ pertenece a la clase C_2

2. En caso contrario, es decir, el n-ésimo miembro del vector de entrenamiento no ha sido correctamente clasificado, el vector de pesos del perceptrón es modificado de acuerdo con la siguiente regla:

$$w(n+1) = w(n) - \eta(n)x(n) \quad \text{si } w^T(n) \cdot x(n) \geq 0 \text{ y } x(n) \text{ pertenece a la clase } C_2$$

y

$$w(n+1) = w(n) + \eta(n)x(n) \quad \text{si } w^T(n) \cdot x(n) < 0 \text{ y } x(n) \text{ pertenece a la clase } C_1$$

donde el parámetro tasa de aprendizaje η(n) controla el ajuste aplicado al vector de pesos en la iteración n.

Si η(n) = η > 0, donde η es una constante independiente del número de iteración n, tenemos una regla de adaptación de incremento fijo para el perceptrón.

A continuación probaremos la convergencia de una regla de adaptación de incremento fijo con η = 1. Claramente, el valor de η no es importante, mientras que sea positivo. Un valor de η ≠ 1 simplemente escala los vectores sin afectar a la separabilidad. El caso de una variable η(n) será considerado más tarde.

La prueba parte de una condición inicial para el vector de pesos: w(0) = 0. Supongamos que $w^T(n) \cdot x(n) < 0$ para n = 1, 2, ... y un vector de entrada x(n) perteneciente al subconjunto X_1. Esto quiere decir que el perceptrón clasifica incorrectamente los vectores x(1), x(2), ... Entonces, con la constante η = 1, podemos escribir lo siguiente:

$$w(n+1) = w(n) + x(n) \text{ para } x(n) \text{ perteneciente a la clase } C_1$$

Dada la condición inicial w(0) = 0, podemos resolver iterativamente esta ecuación para w(n+1), obteniendo el siguiente resultado:

$$w(n+1) = x(1) + x(2) + \ldots + x(n)$$

Ya que las clases C_1 y C_2 son separables linealmente, existe una solución w_0 para la que $w^T x(n) > 0$ para los vectores x(1), ..., x(n) pertenecientes al subconjunto X_1. Para una solución fija w_0, podemos definir entonces un número positivo α por la relación siguiente:

$$\alpha = \min_{x(n) \in X_1} w_0^T x(n)$$

donde $x(n) \in X_1$. Por tanto, multiplicando ambas partes de la ecuación para w(n+1), tenemos que:

$$w_0^T w(n+1) = w_0^T x(1) + w_0^T x(2) + \ldots + w_0^T x(n)$$

Teniendo en cuenta el valor de α definido con anterioridad:

$$w_0^T w(n+1) \geq n\alpha$$

Ahora podemos hace uso de la desigualdad conocida como desigualdad de Cauchy-Schwarz. Dados dos vectores w_0 y w(n+1), la desigualdad de Cauchy-Schwarz establece que:

$$\|w_0\|^2 \|w(n+1)\|^2 \geq \left[w_0^T w(n+1)\right]^2$$

donde $\|\cdot\|$ denota la norma euclidiana y el producto $w_0^T w(n+1)$ es un escalar. Si antes decíamos que $w_0^T w(n+1) \geq n\alpha$, también se cumplirá que $\left[w_0^T w(n+1)\right]^2 \geq n^2\alpha^2$, por lo que tendremos que:

$$\|w_0\|^2 \|w(n+1)\|^2 \geq n^2\alpha^2$$

o equivalentemente:

$$\|w(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|w_0\|^2}$$

Ahora, sigamos otra vía diferente. Vimos que:

$$w(n+1) = w(n) + x(n) \text{ para } x(n) \text{ perteneciente a la clase } C_1$$

Esta ecuación la podemos reescribir de la siguiente manera:

$$w(k+1) = w(k) + x(k) \text{ para k = 1, 2, ..., n y } x(k) \in X_1$$

Entonces, tomando la norma euclidiana y elevando al cuadrado a ambos lados de la ecuación tenemos que:

$$\|w(k+1)\|^2 = \|w(k)\|^2 + \|x(k)\|^2 + 2w^T(k)x(k)$$

Pero, bajo la suposición de que el perceptrón clasifica incorrectamente un vector de entrada x(k) perteneciente al subconjunto X_1, tenemos que $w^T(k)x(k) < 0$. Por tanto, llegamos a que:

$$\|w(k+1)\|^2 \leq \|w(k)\|^2 + \|x(k)\|^2$$

o equivalentemente:

$$\|w(k+1)\|^2 - \|w(k)\|^2 \leq \|x(k)\|^2 \text{ con k = 1, ..., n}$$

Sumando estas desigualdades para k = 1, 2, ..., n, y asumiendo la condición inicial w(0) = 0, llegamos a la siguiente condición:

$$\|w(n+1)\|^2 \leq \sum_{k=1}^{n} \|x(k)\|^2 \leq n\beta$$

donde β es un número positivo definido por:

$$\beta = \max_{x(k) \in X_1} \|x(k)\|^2$$

A partir de estas dos últimas ecuaciones llegamos a que la norma al cuadrado del vector de pesos w(n+1) crece como mucho de forma lineal con el número de iteraciones n.

Claramente, este resultado está en conflicto con otro que obtuvimos con anterioridad, en el cual teníamos que:

$$\|w(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|w_0\|^2}$$

De ambas ecuaciones podemos establecer que n no puede ser mayor que cierto valor n_{max} para el cual se satisfacen ambas ecuaciones con el signo de igualdad. Esto es, n_{max} es la solución de la ecuación:

$$\frac{n_{max}^2\alpha^2}{\|w_0^2\|} = n_{max}^2\beta$$

Con esta ecuación podemos hallar el valor de n_{max} para un determinado vector de pesos w_0 de la siguiente manera:

$$n_{max} = \frac{\beta\|w_0\|^2}{\alpha^2}$$

Por tanto, hemos probado que para η(n) = 1 para todo n, y w(0) = 0, y dado que existe un vector w_0 solución, la regla de adaptación de los pesos sinápticos que conectan las unidades asociativas a la unidad de respuesta del perceptrón debe terminar después de como mucho n_{max} iteraciones. Hacer notar también que por el desarrollo llevado a cabo, pueden existir más de una solución para w_0 o n_{max}.

Ahora, podemos establecer el teorema de convergencia de incremento fijo para un perceptrón monocapa de la siguiente manera:

> *Sean los subconjuntos de vectores de entrenamiento X_1 y X_2 linealmente separables. Sean las entradas presentadas al perceptrón originadas a partir de estos dos subconjuntos. El perceptrón converge después de n_0 iteraciones, en el sentido de que:*
>
> $$w(n_0) = w(n_0 + 1) = w(n_0 + 2) = \ldots$$
>
> *es un vector solución para* $n_0 \leq n_{max}$.

Consideremos ahora el procedimiento de corrección de error absoluto para la adaptación de un perceptrón monocapa para el que η(n) es variable. En particular, sea η(n) el menor número entero para el que:

$$\eta(n)x^T(n)x(n) > \left|w^T(n)x(n)\right|$$

Con este procedimiento encontramos que si el producto $w^T(n)x(n)$ en la iteración n tiene signo incorrecto, entonces $w^T(n+1)x(n)$ en la iteración n+1 tendrá el signo correcto. Esto sugiere que si $w^T(n)x(n)$ tiene signo incorrecto, podemos modificar la secuencia de entrenamiento en la iteración n+1 estableciendo que x(n+1) = x(n). En otras palabras, cada patrón es presentado repetidamente al perceptrón hasta que el patrón es clasificado correctamente.

Hacer notar que el uso de un valor inicial w(0) diferente de 0 simplemente llevará a un aumento o decremento en el número de iteraciones necesarias para la convergencia del algoritmo, dependiendo en cómo se relaciona w(0) con la solución w_0. Entonces, independientemente del valor asignado a w(0), está asegurada la convergencia del perceptrón monocapa.

3.2.1 Resumen del algoritmo de convergencia del perceptrón

a) Variables y parámetros:

$x(n) = \left[-1, x_1(n), x_2(n), \ldots, x_p(n)\right]^T$

$w(n) = \left[\theta(n), w_1(n), w_2(n), \ldots, w_p(n)\right]^T$

$\theta(n)$ es el umbral

$y(n)$ es la respuesta actual

$d(n)$ es la respuesta deseada

η es el parámetro tasa de aprendizaje, constante positiva menor que la unidad

b) Paso 1: Inicialización

Se establece w(0) = 0 y se llevan a cabo los siguientes cálculos para los instantes de tiempo n = 1, 2, ...

c) Paso 2: Activación

En el instante n, se activa el perceptrón aplicando el vector de entrada x(n) y la respuesta deseada d(n).

d) Paso 3: Cálculo de la respuesta actual

Se calcula la respuesta actual del perceptrón de la siguiente manera:

$$y(n) = \text{sgn}\left[w^T(n)x(n)\right]$$

donde sgn (·) es la función signo.

e) Paso 4: Adaptación del vector de pesos

Se actualiza el vector de pesos del perceptrón de la siguiente manera:

$$w(n+1) = w(n) + \eta\left[d(n) - y(n)\right]x(n)$$

donde

$$d(n) = \begin{cases} +1 & \text{si x(n) pertenece a la clase } \mathrm{C}_1 \\ -1 & \text{si x(n) pertenece a la clase } \mathrm{C}_2 \end{cases}$$

f) Paso 5

Se incrementa n en una unidad y se vuelve al paso 2

Como se puede ver, la adaptación del vector de pesos w(n) es realizada en forma de regla de corrección de error, como se muestra a continuación:

$$w(n+1) = w(n) + \eta[d(n) - y(n)]x(n)$$

donde η es el parámetro tasa de aprendizaje, y la diferencia d(n)-y(n) juega el papel de la señal de error. El parámetro tasa de aprendizaje es una constante positiva limitada al rango $0 < \eta \leq 1$. A la hora de asignarle un valor en este rango, debemos tener en cuenta dos cosas:

- Promediado de entradas pasadas para proporcionar estimaciones de los pesos estables, lo que requiere un valor pequeño para η.

- Rápida adaptación con respecto a los cambios reales en las distribuciones subyacentes del proceso responsable de la generación del vector de entrada x, lo que requiere un valor grande para η.

3.3 Medida del rendimiento

En el apartado anterior hemos desarrollado el algoritmo de convergencia del perceptrón sin hacer ninguna referencia a la medida del rendimiento. ¿Cual es una medida apropiada del rendimiento para un perceptrón monocapa?. Una respuesta obvia sería un promedio de la probabilidad de clasificación incorrecta, definida como el promedio de la probabilidad de que el perceptrón realice una decisión en favor de una clase en particular cuando el vector de entrada ha sido obtenido de otra clase. Desafortunadamente, una medida del rendimiento de este tipo no deriva analíticamente a un algoritmo de aprendizaje. Shynk propuso el uso de una función de rendimiento que beneficia el funcionamiento de un perceptrón:

$$J = -E[e(n)v(n)]$$

donde e(n) es la señal de error definida como la diferencia entre la respuesta deseada d(n) y la respuesta actual y(n) del perceptrón, y v(n) es la salida del combinador lineal. Hacer notar que la señal de error e(n) no es una señal cuantizada, pero sí la diferencia entre dos señales cuantizadas. La estimación instantánea de la función de rendimiento es la función variante en el tiempo siguiente:

$$J(n) = -e(n)v(n) = -[d(n) - y(n)]v(n)$$

El vector gradiente instantáneo se define como la derivada de la estimación $\hat{J}(n)$ con respecto al vector de pesos w(n), como se muestra a continuación:

$$\nabla_w \hat{J}(n) = \frac{\partial J(n)}{\partial w(n)}$$

Entonces, reconociendo que la respuesta deseada d(n) es independiente de w(n) y el hecho de que la respuesta actual y(n) tiene un valor constante igual a +1 ó –1, llegamos a que:

$$\nabla_w \hat{J}(n) = -[d(n) - y(n)]\frac{\partial v(n)}{\partial w(n)}$$

Ya vimos que v(n) se definía de la siguiente manera: $v(n) = w^T(n)x(n)$. Por lo que se llega a que:

$$\frac{\partial v(n)}{\partial w(n)} = x(n)$$

Visto esto, el vector gradiente instantáneo queda de la siguiente manera:

$$\nabla_w \hat{J}(n) = -[d(n) - y(n)] \cdot x(n)$$

Y de acuerdo con la regla de corrección de error descrita en el capítulo 2, podemos expresar ahora el cambio aplicado al vector de pesos de la siguiente manera:

$$\Delta w(n) = -\eta \nabla_w \hat{J}(n) = \eta[d(n) - y(n)]x(n)$$

donde η es el parámetro tasa de aprendizaje. Ésta es precisamente la corrección realizada al vector de pesos al pasar de la iteración n a la iteración n+1. Acabamos de demostrar que la función de rendimiento instantáneo $\hat{J}(n)$ recién hallada es verdaderamente la medida de rendimiento correcta para un perceptrón monocapa.

4 PERCEPTRONES MULTICAPA

4.1 Introducción

En este apartado vamos a estudiar una clase importante de redes neuronales, denominadas redes feedforward multicapa. Estas redes constan de unos nodos de entrada, que constituyen la capa de entrada, una o más capas ocultas compuestas de nodos de computación, y una capa de salida. Estas redes neuronales son conocidas normalmente como perceptrones multicapa, y representan una generalización del perceptrón monocapa estudiado en el apartado anterior.

Los perceptrones multicapa se han aplicado con éxito para resolver distintos problemas entrenándolos en modo supervisado con el algoritmo de propagación hacia atrás del error (Backpropagation). Este algoritmo consta de dos pasadas a través de las diferentes capas de la red: una pasada hacia adelante y otra hacia atrás. En la pasada hacia adelante, el patrón de actividad (vector de entrada) es aplicado a los nodos de entrada de la red, y sus efectos se propagan a través de la red, capa por capa. Finalmente, la red produce la salida como respuesta al vector de entrada. Durante la pasada hacia adelante, los pesos sinápticos de la red permanecen constantes, mientras que en la pasada hacia atrás, los pesos sinápticos son ajustados de acuerdo a la regla de corrección del error. La salida actual se resta a la salida deseada para obtener una señal de error. Esta señal de error se propaga hacia atrás por la red, ajustándose los pesos sinápticos de forma que la salida de la red se aproxime a la salida deseada.

Un perceptrón multicapa tiene tres características distintivas:

1. El modelo de cada neurona en la red incluye una no linealidad a la salida. Esta no linealidad es suave (diferenciable), y suele usarse para ello la función sigmoidal:

$$y_j = \frac{1}{1+e^{-v_j}}$$

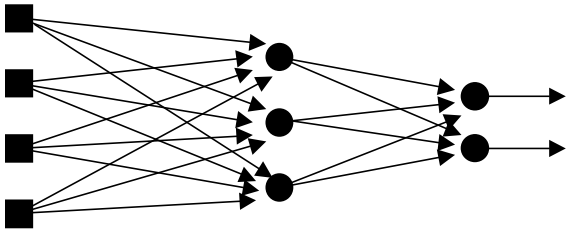
   donde v_j es el nivel de actividad interna de la neurona j, e y_j es la salida de la neurona. La presencia de la no linealidad es importante porque, de otra forma, la relación entrada-salida de la red podría reducirse a la de un perceptrón monocapa.

2. La red contiene una o más capas de neuronas ocultas que no son parte ni de la entrada ni de la salida. Estas neuronas ocultas permiten a la red aprender tareas complejas extrayendo progresivamente las características más significativas de los patrones de entrada.

3. La red presenta un alto grado de conectividad, determinado por los enlaces sinápticos.

Es a través de esta combinación de características junto con la habilidad de aprender de la experiencia a través del entrenamiento lo que hace que el perceptrón obtenga su poder de cómputo.

En la siguiente figura se muestra un perceptrón multicapa con una capa oculta:



Esta red está completamente conectada, lo que significa que una neurona de cualquier capa está conectada con todas las neuronas de la capa precedente.

Cada neurona oculta o neurona de salida del perceptrón multicapa está diseñado para realizar dos cálculos:

1. El cómputo de la señal de salida de una neurona, la cual se expresa como una función no lineal continua de las señales de entrada y pesos sinápticos asociados con esa neurona.

2. El cómputo de una estimación instantánea del vector gradiente (gradiente de la superficie de error respecto a los pesos sinápticos conectados a la entrada de una neurona), necesario para la pasada hacia atrás a través de la red.

4.2 Algoritmo Backpropagation

La señal de error a la salida de la neurona j en el instante de tiempo n está definida por:

$$e_j(n) = d_j(n) - y_j(n)$$

siendo la neurona j un nodo de salida.

Definimos el valor instantáneo del error cuadrático para la neurona j como $\frac{1}{2}e_j^2(n)$. Correspondientemente, el valor instantáneo de la suma de los errores cuadráticos es obtenido sumando ese valor sobre todas las neuronas de la capa de salida; éstas son las únicas neuronas visibles para las que

se puede calcular la señal de error. La suma instantánea de los errores cuadráticos de la red se puede escribir, por tanto, de la siguiente manera:

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

donde el conjunto C incluye todas las neuronas de la capa de salida de la red. Sea N el número total de patrones contenidos en el conjunto de entrenamiento. El error cuadrático medio es obtenido sumando $\xi(n)$ para todo n y normalizando con respecto a N, como se muestra a continuación:

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n)$$

La suma instantánea de los errores cuadráticos $\xi(n)$, y por tanto el error cuadrático medio ξ_{av}, es función de todos los parámetros libres de la red. Para un conjunto de entrenamiento dado, ξ_{av} representa la función de coste como medida del funcionamiento del aprendizaje del conjunto de entrenamiento. El objetivo del proceso de aprendizaje es ajustar los parámetros libres de la red de forma que se minimice ξ_{av}. Para realizar esto, consideraremos un método de entrenamiento en el que los pesos son actualizados patrón a patrón. El ajuste de los pesos se realiza de acuerdo con los respectivos errores calculados para cada patrón presentado a la red. La media aritmética de estos cambios individuales de los pesos a lo largo del conjunto de entrenamiento es por tanto una estimación del verdadero cambio que resultaría de la modificación de los pesos en base a la minimización de la función de coste ξ_{av} a lo largo del conjunto de entrenamiento.

Como ya vimos, el nivel de actividad interna de una neurona j a la entrada de la no linealidad se puede expresar de la siguiente manera:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n)$$

donde p es el número total de entradas (sin incluir el umbral) aplicadas a la neurona j. El peso sináptico w_{j0} (correspondiente a la entrada constante $y_0 = -1$) se corresponde con el valor del umbral θ_j aplicado a la neurona j. La señal $y_j(n)$ que aparece a la salida de la neurona j en la iteración n es:

$$y_j(n) = \varphi_j\left(v_j(n)\right)$$

El algoritmo Backpropagation aplica una corrección $\Delta w_{ji}(n)$ al peso sináptico $w_{ji}(n)$, que es proporcional al gradiente instantáneo, expresado de la siguiente manera:

$$\Delta w_{ji}(n) = \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

De acuerdo con la regla de la cadena, podemos expresar este gradiente como:

$$\Delta w_{ji}(n) = \frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Este gradiente representa un factor de sensibilidad, determinando la dirección de búsqueda en el espacio de pesos para el peso sináptico w_{ji}.

Si derivamos la expresión de la suma de los errores cuadráticos respecto a $e_j(n)$, obtenemos que:

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n)$$

Así mismo, como la señal de error se calcula como la salida deseada menos la salida actual, se tiene que:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

También, como $y_j(n)$ dijimos que era una función φ dependiente de $v_j(n)$, la derivada de la salida actual respecto al nivel de actividad interna queda como:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'\left(v_j(n)\right)$$

donde el uso de la prima significa derivada respecto al argumento. Finalmente, a partir de la definición de nivel de actividad interna de una neurona se llega a que:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

Haciendo uso de todas estas expresiones que hemos ido viendo y sustituyéndolas en la fórmula de la corrección aplicada a los pesos sinápticos tenemos que:

$$\Delta w_{ji}(n) = \frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))\, y_i(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

Esta expresión es conocida como la regla de la delta, y el parámetro η es una constante que determina la tasa de aprendizaje; se denomina parámetro tasa de aprendizaje del algoritmo Backpropagation. Otra forma de expresar esta corrección se muestra a continuación:

$$\Delta w_{ji}(n) = \eta \partial_j(n)\, y_i(n)$$

donde el gradiente local $\delta_j(n)$ esta a su vez definido por:

$$\partial_j(n) = -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$$

Este gradiente local apunta hacia la dirección en la que deben cambiar los pesos sinápticos. De acuerdo con la expresión deducida, el gradiente local $\delta_j(n)$ para la neurona de salida j es igual al producto de la señal de error $e_j(n)$ y la derivada de la función de activación asociada.

Un factor importante en el cálculo del ajuste aplicado a los pesos sinápticos es la señal de error a la salida de la neurona j. En este contexto, podemos identificar dos casos distintos. En el primer caso, la neurona j es un nodo de salida. Este caso es simple de tratar, ya que a cada nodo de salida se le proporciona una salida deseada, por lo que se puede calcular la señal de error directamente. En el segundo caso, la neurona j es un nodo oculto. Aunque estas neuronas no son directamente accesibles, comparten responsabilidad en cuanto al error obtenido a la salida de la red.

4.2.1 Caso I: La neurona j es un nodo de salida

Cuando la neurona j se encuentra en la capa de salida, se le suministra la salida deseada. Entonces, la señal de error se puede calcular de la siguiente manera:

$$e_j(n) = d_j(n) - y_j(n)$$

Una vez obtenida la señal de error, se puede calcular fácilmente el gradiente local $\delta_j(n)$ mediante la siguiente expresión:

$$\partial_j(n) = -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$$

4.2.2 Caso II: La neurona j es un nodo oculto

Cuando la neurona j se encuentra en una capa oculta, no hay una salida deseada específica para esa neurona. La señal de error para una neurona oculta tendría que ser determinada recursivamente basándose en las señales de error de todas las neuronas a las que la neurona oculta se encuentra directamente conectada; es aquí donde se complica el algoritmo. Consideremos una neurona j perteneciente a una capa oculta. El gradiente local se puede redefinir de la siguiente manera para la neurona oculta j:

$$\partial_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \xi(n)}{\partial y_j} \varphi'_j(v_j(n))$$

Para calcular la derivada parcial $\frac{\partial \xi(n)}{\partial y_j(n)}$ podemos proceder de la siguiente manera:

$$\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

Si derivamos esta ecuación respecto a la señal función $y_j(n)$, y aplicando la regla de la cadena, obtenemos:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_j(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

Además, sabemos que:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n))$$

por lo que:

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n))$$

También, sabemos que el nivel de actividad interna de una neurona k se puede expresar de la siguiente manera:

$$v_k(n) = \sum_{i=0}^{q} w_{kj}(n) y_j(n)$$

donde q es el número total de entradas (excluyendo el umbral) aplicado a la neurona k. Aquí, de nuevo, el peso sináptico $w_{k0}(n)$ se corresponde con el umbral $\theta_k(n)$ aplicado a la neurona k, y la correspondiente entrada y_0 está fijada al valor –1. Si derivamos esta ecuación respecto a $y_j(n)$ llegamos a que:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

Si utilizamos estas dos últimas expresiones obtenidas, podemos reformular la derivada parcial deseada de la siguiente manera:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi_k'(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n)$$

Finalmente, si introducimos esta expresión en la definición del gradiente local $\delta_j(n)$, obtenemos que:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

El factor $\varphi'_j(v_j(n))$ implicado en el cálculo del gradiente local depende exclusivamente de la función de activación asociada con la neurona oculta j. El otro factor implicado, la suma sobre k, depende de dos conjuntos de términos. El primero de ellos, $\delta_k(n)$, requiere el conocimiento de las señales de error $e_k(n)$ para todas aquellas neuronas que se encuentran en la capa inmediatamente a la derecha de la neurona j, y que están directamente conectadas a la neurona j. El segundo de ellos, $w_{kj}(n)$, consiste en los pesos sinápticos asociados con estas conexiones.

Resumamos las relaciones obtenidas para el algoritmo Backpropagation. Primero, la corrección $\Delta w_{ji}(n)$ aplicada al peso sináptico que conecta la neurona i a la neurona j viene definida por la regla de la delta:

$$\begin{pmatrix} \text{correción} \\ \text{del peso} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{parámetro} \\ \text{tasa de aprendizaje} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{gradiente} \\ \text{local} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{señal de entrada} \\ \text{de la neurona j} \\ y_i(n) \end{pmatrix}$$

En segundo lugar, el gradiente local $\delta_j(n)$ depende de si la neurona j es un nodo de salida o es un nodo oculto:

1. Si la neurona j es un nodo de salida, $\delta_j(n)$ es igual al producto de la derivada $\varphi'_j(v_j(n))$ y la señal de error $e_j(n)$, ambas asociadas a la neurona j.

2. Si la neurona j es un nodo oculto, $\delta_j(n)$ es igual al producto de la derivada $\varphi'_j(v_j(n))$ asociada y la suma ponderada de las deltas calculadas para las neuronas en la siguiente capa oculta o de salida que esté conectada a la neurona j.

4.2.3 Las dos pasadas del cálculo

En la aplicación del algoritmo Backpropagation, se pueden distinguir dos pasadas en el cálculo, una primera pasada hacia adelante y otra hacia atrás.

En la pasada hacia adelante los pesos sinápticos permanecen inalterados, y las señales de salida de las neuronas son calculadas neurona a neurona. Concretamente, la señal a la salida de la neurona j es calculada como:

$$y_j(n) = \varphi(v_j(n))$$

donde $v_j(n)$ es el nivel de actividad interna de la neurona j, definido como:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n)$$

donde p es el número total de entradas (sin incluir el umbral) aplicadas a la neurona j y w_{ji} es el peso sináptico que conecta la neurona i a la j, e $y_i(n)$ es la señal de entrada de la neurona j. Si la neurona j se encuentra en la primera capa oculta de la red, entonces el índice i se refiere al i-ésimo terminal de la entrada de la red, por lo que podemos escribir:

$$y_i(n) = x_i(n)$$

donde $x_i(n)$ es el i-ésimo elemento del vector de entrada. Si, en cambio, la neurona j pertenece a la capa de salida de la red, el índice j se refiere al j-ésimo terminal de salida de la red, por lo que podemos escribir:

$$y_j(n) = o_j(n)$$

donde o_j es el j-ésimo elemento del vector de salida. Esta salida es comparada con la salida deseada $d_j(n)$, obteniendo la señal de error $e_j(n)$ para la j-ésima neurona de salida. Por ello es que la fase hacia adelante de cálculo comience con la primera capa presentándosele el vector de entrada, y termine en la capa de salida calculándose la señal de error para cada neurona de esta capa.

La pasada hacia atrás, en cambio, comienza en la capa de salida pasando las señales de error hacia la izquierda a través de la red, capa por capa, y calculando recursivamente el gradiente local para cada neurona. Este proceso recursivo permite que los pesos sinápticos sufran cambios de acuerdo con la regla de la delta. Para una neurona localizada en la capa de salida, δ es simplemente la señal de error de esa neurona multiplicada por la primera derivada de su no linealidad. Halladas estas δ podemos determinar los cambios a aplicar a los pesos sinápticos correspondientes a todas las neuronas de la capa de salida. Entonces pasamos a calcular las deltas para todas las neuronas de la penúltima capa y los cambios en los pesos sinápticos de las conexiones que las alimentan. Este cálculo recursivo continua capa por capa propagando los cambios realizados a los pesos sinápticos.

4.2.4 No linealidad sigmoidal

El cálculo de la δ para cada neurona del perceptrón multicapa requiere el conocimiento de la derivada de la función de activación $\varphi(\cdot)$ asociada con dicha neurona. Para que exista esta derivada, es necesario que esa función sea continua. La diferenciabilidad es el único requisito que la función de activación debe cumplir. Un ejemplo de función de activación no lineal continuamente diferenciable comúnmente usada en perceptrones multicapa es la no linealidad sigmoidal. Una forma particular de ésta es definida para la neurona j por la función logística:

$$y_j(n) = \varphi_j(v_j(n)) = \frac{1}{1+e^{-v_j(n)}}; \qquad -\infty < v_j(n) < +\infty$$

donde $v_j(n)$ es el nivel de actividad interna de la neurona j. De acuerdo con esta no linealidad, la amplitud de la salida cae en el rango $0 \le y_j \le 1$. Otro tipo de no linealidad sigmoidal es la tangente hiperbólica, la cual es antisimétrica respecto al origen y cuya amplitud cae en el rango $-1 \le y_j \le +1$.

Fijémonos en la función logística. Si diferenciamos ambos lados de la ecuación antes presentada tenemos que:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n)) = \frac{e^{-v_j(n)}}{\left(1+e^{-v_j(n)}\right)^2}$$

Podemos utilizar la propia definición de la función logística para eliminar el término exponencial de esta expresión, con lo que obtenemos que:

$$\varphi'_j(v_j(n)) = y_j(n)\left[1 - y_j(n)\right]$$

Para una neurona j localizada en la capa de salida, $y_j(n) = o_j(n)$. Entonces, podemos expresar el gradiente local para la neurona j como:

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)) = \left[d_j(n) - o_j(n)\right] \cdot o_j(n) \cdot \left[1 - o_j(n)\right]$$

donde $o_j(n)$ es la señal de salida de la neurona j, y $d_j(n)$ es la respuesta deseada para ella. En cambio, para una neurona oculta arbitraria j, podemos expresar el gradiente local como:

$$\delta_j(n) = \varphi_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) = y_j(n)\left[1 - y_j(n)\right] \cdot \sum_k \delta_k(n) w_{kj}(n)$$

Hacer notar que la derivada de la función de activación logística alcanza su mayor valor cuando $y_j(n) = 0.5$, y su mínimo valor (cero) cuando $y_j(n) = 0$, o $y_j(n) = 1$. Como la magnitud del cambio en un peso sináptico de la red es proporcional a $\varphi'_j(v_j(n))$, se sigue que para una función de activación sigmoidal los pesos sinápticos se ven más modificados para aquellas neuronas de la red cuyas señales de salida se encuentran en un rango intermedio. Es esta característica la que contribuye a la estabilidad de este algoritmo.

4.2.5 Tasa de aprendizaje

El algoritmo Backpropagation proporciona una aproximación a la trayectoria en el espacio de pesos calculada por el método del mayor descenso. Cuanto más pequeña hagamos el parámetro tasa de aprendizaje η, menores serán los cambios realizados a los pesos sinápticos de una iteración a otra, y más suave será la trayectoria en el espacio de pesos. En cambio, si hacemos más grande el valor del parámetro tasa de aprendizaje para acelerar el proceso de aprendizaje, los grandes cambios realizados en los pesos sinápticos pueden hacer la red inestable. Un método muy simple para incrementar la tasa de aprendizaje evitando el peligro de la inestabilidad es modificar la regla de la delta incluyendo un término momento, como se muestra a continuación:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

donde α es normalmente un número positivo llamado constante de momento, que controla un bucle de realimentación. Esta expresión se denomina regla de la delta generalizada, e incluye la regla de la delta como un caso particular (α=0).

Para ver los efectos de la presentación de la secuencia de patrones en los pesos sinápticos debido a la constante de momento α, podemos reescribir la expresión para el cálculo del cambio aplicado a los pesos sinápticos como una serie temporal, donde el índice t va desde el instante inicial t hasta el instante actual n. La ecuación puede verse como una ecuación diferencial de primer orden, que tras resolverse queda de la siguiente manera:

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$$

De las expresiones presentadas en la derivación del algoritmo, sabemos que:

$$\delta_j(n) y_i(n) = -\frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

por lo que podemos reescribir la expresión de $\Delta w_{ji}(n)$ de la siguiente manera:

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \xi(t)}{\partial w_{ji}(t)}$$

Basándonos en esta relación, podemos hacer las siguientes observaciones:

1. El ajuste $\Delta w_{ji}(n)$ representa la suma de una serie temporal pesada y exponencial. Para que la serie sea convergente, la constante de momento debe ser restringida al rango $0 \leq | \alpha | < 1$. Cuando α es cero, el algoritmo Backpropagation trabaja sin momento.

2. Cuando la derivada parcial $\partial\xi(t)/\partial w_{ji}(t)$ tiene el mismo signo en iteraciones consecutivas, la suma pesada exponencialmente $\Delta w_{ji}(n)$ crece en magnitud, y así el peso $w_{ji}(n)$ se ve modificado en un grado mayor. Aquí, la inclusión del término momento del que hablamos antes en el algoritmo Backpropagation hace que el algoritmo se acelere.

3. Cuando la derivada parcial $\partial\xi(t)/\partial w_{ji}(t)$ tiene distinto signo en iteraciones consecutivas, la suma pesada exponencialmente $\Delta w_{ji}(n)$ decrece en magnitud, y así el peso $w_{ji}(n)$ se ve ajustado en una pequeña cantidad. Aquí, la inclusión del término momento tiene un efecto estabilizante.

Por tanto, la inclusión del término momento en el algoritmo Backpropagation representa una modificación pequeña en la actualización de los pesos, pero puede tener efectos beneficiosos en el comportamiento de aprendizaje del algoritmo. Además, el momento puede prevenir al proceso de aprendizaje de terminar en un mínimo local de la superficie de error.

4.2.6 Modos de entrenamiento

En una aplicación práctica del algoritmo Backpropagation, el aprendizaje se realiza mediante las muchas presentaciones de un determinado conjunto de ejemplos de entrenamiento al perceptrón multicapa. Una presentación completa del conjunto de entrenamiento durante el proceso de aprendizaje se denomina una época. El proceso de aprendizaje se realiza época por época hasta que los pesos sinápticos y los umbrales de la red se estabilizan y el error cuadrático medio sobre el conjunto completo de entrenamiento converge a cierto valor mínimo. Resulta aconsejable aleatorizar el orden de presentación de los ejemplos de entrenamiento de una época a la otra. Esta aleatorización tiende a hacer la búsqueda en el espacio de pesos de manera estocástica a lo largo de los ciclos de aprendizaje, evitando por tanto la posibilidad de ciclos en la evolución de los vectores de pesos sinápticos.

Para un conjunto de entrenamiento dado, el aprendizaje se puede llevar a cabo básicamente mediante dos métodos:

1. Modo patrón. En el modo patrón del aprendizaje con el algoritmo Backpropagation, la actualización de los pesos es llevada a cabo después de la presentación de cada uno de los ejemplos de entrenamiento; éste es en realidad el modo mediante el cual se ha obtenido el algoritmo. Consideremos una época consistente en N ejemplos de entrenamiento (patrones) dispuestos en el orden [x(1),d(1)], ..., [x(N), d(N)]. El primer ejemplo [x(1),d(1)] de la época es presentado a la red, y los cálculos hacia adelante y hacia atrás descritos con anterioridad son llevados a cabo, resultando en ciertos ajustes a los pesos sinápticos y umbrales de la red. Entonces, el segundo ejemplo [x(2),d(2)] de la época es presentado, y se repite el proceso de cálculo, llevando a cabo nuevos ajustes en los pesos sinápticos y umbrales. Este proceso se repite hasta el último ejemplo [x(N),d(N)]. Denotemos por $\Delta w_{ji}(n)$ al cambio aplicado al peso sináptico w_{ji} después de la presentación del patrón n. Entonces, el cambio del peso de la red $\Delta\lfloor_{ji}$, promediado sobre el conjunto completo de entrenamiento de N patrones, viene dado por:

$$\Delta \hat{w}_{ji} = \frac{1}{N}\sum_{n=1}^{N}\Delta w_{ji}(n) = -\frac{\eta}{N}\sum_{n=1}^{N}\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -\frac{\eta}{N}\sum_{n=1}^{N} e_j(n)\frac{\partial e_j(n)}{\partial w_{ji}(n)}$$

2. Modo batch. En este modo, el ajuste de los pesos se lleva a cabo después de la presentación de todos los ejemplos de entrenamiento que constituyen una época. Para una época en particular, definimos la función de coste como la suma de los errores cuadráticos:

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n)$$

donde la señal de error $e_j(n)$ pertenece a la neurona de salida j para el ejemplo de entrenamiento n, y que viene definida como la diferencia entre la salida deseada y la salida actual para esa neurona. En esta expresión, la suma sobre j se realiza para todas las neuronas pertenecientes a la capa de salida de la red, mientras que la suma sobre n se realiza para todos los ejemplos de entrenamiento del conjunto en la época en cuestión. Para un parámetro tasa de aprendizaje η, el ajuste aplicado al peso sináptico w_{ji}, que conecta la neurona i a la neurona j, viene definido por la regla de la delta:

$$\Delta w_{ji} = -\eta\frac{\partial \xi_{av}}{\partial w_{ji}} = -\frac{\eta}{N}\sum_{n=1}^{N} e_j(n)\frac{\partial e_j(n)}{\partial w_{ji}}$$

Se puede ver que el cambio medio aplicado a los pesos en el modo patrón es diferente al valor correspondiente obtenido para el caso del modo batch, presumiblemente para la misma reducción en el error cuadrático medio que resulta de la presentación de un conjunto de entrenamiento completo. En realidad, $\Delta\llcorner_{ji}$ para el modo patrón a patrón es una estimación de Δw_{ji} para el modo batch.

Desde un punto de vista operacional , se prefiere el modo patrón al modo batch por la menor necesidad de almacenamiento necesaria para cada enlace sináptico. Además, suponiendo que los patrones son presentados a la red en un orden aleatorio, el uso del ajuste patrón a patrón hace que la búsqueda en el espacio de pesos sea de naturaleza estocástica, lo que hace que el algoritmo sea menos propenso a quedarse atrapado en un mínimo local. En cambio, el uso del modo batch proporciona una mejor estimación del vector gradiente. Finalmente, la efectividad de ambos métodos de entrenamiento dependerá de la aplicación en uso.

4.2.7 Criterio de parada

En general, no se puede demostrar que el algoritmo Backpropagation converja, así como no hay un criterio bien definido para parar su funcionamiento. Sí que hay algunos criterios razonables que pueden ser usados para terminar el ajuste de los pesos. Para formular un criterio de este tipo, hay que pensar en términos de propiedades únicas de un mínimo local o global de la superficie de error. Sea el vector de pesos w* un mínimo, que puede ser local o global. Una condición necesaria para que w* sea un mínimo

es que el vector de gradiente g(w) de la superficie de error respecto al vector de pesos w sea cero cuando w=w*. Acorde con esto, podemos formular un criterio de convergencia sensible para el algoritmo de aprendizaje Backpropagation de la siguiente manera:

- Se considera que el algoritmo Backpropagation ha convergido cuando la norma euclidiana del vector de gradiente alcanza un gradiente umbral suficientemente pequeño.

El inconveniente de este criterio de convergencia es que, para pruebas satisfactorias, el tiempo de aprendizaje puede ser muy largo. Además, requiere el cálculo del vector de gradiente g(w).

Otra propiedad única de un mínimo que podemos usar es que la función de coste o medida del error $\xi_{av}(w)$ es estacionaria en el punto w=w*. Podemos entonces sugerir un criterio de convergencia diferente:

- Se considera que el algoritmo Backpropagation ha convergido cuando la tasa absoluta de cambio en el promedio del error cuadrático por época es suficientemente pequeño.

Típicamente, la tasa de cambio en el promedio del error cuadrático se considera suficientemente pequeña si cae en el rango de 0.1 a 1 por ciento por época; a veces se usa un valor cercano al 0.01 por ciento por época.

Una variación de este segundo criterio de convergencia del algoritmo es requerir que el máximo valor del error cuadrático promedio $\xi_{av}(w)$ sea igual o menor que un umbral suficientemente pequeño. Este criterio podría ser enunciado de la siguiente manera:

- El algoritmo Backpropagation ha terminado en el vector de pesos w_{final} cuando $\| g(w_{final}) \| \leq \varepsilon$, donde ε es un gradiente umbral suficientemente pequeño, o $\xi_{av}(w_{final}) \leq \tau$, donde τ es un umbral de energía de error suficientemente pequeño.

Otro criterio de convergencia útil sería probar el grado de generalización de la red después de cada iteración del aprendizaje. El proceso de aprendizaje acabaría cuando el grado de generalización fuese adecuado o fuese aparente que hubiese alcanzado su valor máximo.

4.3 Inicialización

El primer paso en el aprendizaje por propagación hacia atrás es, por supuesto, inicializar la red. Una buena elección en los valores iniciales de los parámetros libres de la red puede ser de tremenda ayuda. En los casos en los que exista información a priori, puede ser mejor usar la información a priori para

averiguar los valores iniciales de los parámetros libres. Si no se dispone de información a priori, lo usual es inicializar estos parámetros libres a valores aleatorios uniformemente distribuidos dentro de un rango pequeño de valores.

Una elección desacertada de los pesos iniciales puede llevar a un fenómeno conocido como saturación prematura. Este fenómeno se refiere a la situación en la que la suma instantánea de errores cuadráticos $\xi(n)$ se mantiene prácticamente constante durante cierto tiempo en el proceso de aprendizaje. Un fenómeno de este tipo no puede ser considerado un mínimo local, ya que el error cuadrático continua decreciendo una vez que este periodo finaliza.

Cuando se aplica un patrón de entrenamiento a la capa de entrada de un perceptrón multicapa, los valores de salida de la red son calculados mediante un secuencia de cálculos hacia adelante que implican productos internos y transformaciones sigmoidales. Esto es seguido por una secuencia de cálculos hacia atrás que implican el cálculo de señales de error y de la pendiente correspondiente a la función de activación sigmoidal, concluyendo en el ajuste de los pesos sinápticos. Supongamos que, para un patrón de entrenamiento en particular, el nivel de actividad interna de una neurona de salida alcanza un valor elevado. Entonces, asumiendo que la función de activación sigmoidal de la neurona tiene los valores límites -1 y $+1$, encontramos que la correspondiente pendiente de la función de activación para esa neurona será muy pequeña, y el valor de la salida de esa neurona será muy próximo a -1 ó $+1$. En tal caso, decimos que esa neurona está en saturación. Si el valor de salida se encuentra cercano a $+1$ cuando el valor objetivo es -1, o viceversa, decimos que la neurona está incorrectamente saturada. Cuando esto ocurre, el ajuste aplicado a los pesos sinápticos de la neurona serán pequeños, y la red puede tardar mucho tiempo en salir de ahí.

En la fase inicial del aprendizaje por propagación hacia atrás pueden existir tanto neuronas no saturadas como incorrectamente saturadas en la capa de salida de la red. Mientras que el proceso de aprendizaje continua, los pesos sinápticos asociados con las neuronas de salida no saturadas cambian rápidamente, ya que las señales de error y gradientes correspondientes toman valores relativamente elevados, por lo que resulta en una reducción de la suma instantánea de los errores cuadráticos $\xi(n)$. Si, en este punto, las neuronas de salida incorrectamente saturadas permanecen saturadas para determinados patrones de entrenamiento, entonces puede aparecer el fenómeno de la saturación prematura, con $\xi(n)$ permaneciendo prácticamente constante.

Existe una fórmula para la probabilidad de saturación prematura en el aprendizaje por propagación hacia atrás derivada del modo batch de actualización, y que ha sido verificada usando simulaciones por computador. La esencia de esta fórmula puede ser resumida de la siguiente forma:

1. La saturación incorrecta es evitada escogiendo que los valores de los pesos sinápticos y umbrales de la red sean valores aleatorios uniformemente distribuidos de un pequeño rango de valores.

2. La saturación incorrecta es menos propensa a ocurrir cuando el número de neuronas ocultas es pequeño, consistente con un funcionamiento satisfactorio de la red.

3. La saturación incorrecta raramente ocurre cuando las neuronas de la red operan en su región lineal.

4.4 Aproximación de funciones

Un perceptrón multicapa entrenado con el algoritmo Backpropagation puede ser visto como una manera práctica de realizar un mapeado no lineal entre entradas y salidas de naturaleza general. Denotemos por p el número de nodos de entrada de un perceptrón multicapa y denotemos por q el número de neuronas en la capa de salida de la red. La relación entrada-salida de la red define un mapeado de un espacio de entrada euclidiano de dimensión p a un espacio de salida euclidiano de dimensión q, que es continuamente diferenciable. La pregunta que se plantea es: ¿cuál es el número mínimo de capas ocultas en un perceptrón multicapa con un mapeado entrada-salida que proporcione una realización aproximada de cualquier mapeado continuo?.

4.4.1 Teorema de la Aproximación Universal

Numerosos estudios han demostrado que una red feedforward con una sola capa oculta puede proporcionar una aproximación de cualquier función continua.

Sea $\varphi(\cdot)$ una función no constante, limitada, continua y monótona creciente. Sea I_p el hipercubo unidad de dimensión p $[0,1]^p$. El espacio de funciones continuas en I_p es denotado por $C(I_p)$. Entonces, dada cualquier función $f \in C(I_p)$ y $\varepsilon>0$, existe un entero M y conjuntos de constantes reales α_i, θ_i, y w_{ij}, donde i = 1, ..., M y j = 1, ..., p de forma que podemos definir:

$$F(x_1,...x_p) = \sum_{i=1}^{M} \alpha_i \varphi\left(\sum_{j=1}^{p} w_{ij} x_j - \theta_i \right)$$

como una realización aproximada de la función f (·); esto es:

$$\left| F(x_1,...,x_p) - f(x_1,...,x_p) < \varepsilon \right|$$

parra todo $\{x_1, ..., x_p\} \in I_p$.

Este teorema es directamente aplicable a perceptrones multicapa. Hagamos notar primero que la función logística $1/[1+e^{-v}]$ usada como no linealidad en el modelo de neurona para la construcción de un perceptrón multicapa es en realidad una función no constante, limitada y monótona creciente; por tanto, satisface las condiciones impuestas a la función $\varphi(\cdot)$. La expresión expuesta como realización aproximada de una función representa la salida de un perceptrón multicapa que describimos a continuación:

1. La red tiene p nodos de entrada y una única capa oculta constituida por M neuronas; las entradas son denotadas por $x_1, ..., x_p$.

2. La neurona oculta i tiene pesos sinápticos $w_{i1}, ..., w_{ip}$ y umbral θ_i.

3. La salida de la red es una combinación lineal de las salidas de las neuronas ocultas, con $\alpha_1, ..., \alpha_M$ definiendo los coeficientes de dicha combinación.

El teorema de la aproximación universal es un teorema de existencia en el sentido de que proporciona la justificación matemática para la aproximación de una función continua arbitraria. En efecto, el teorema establece que una única capa oculta es suficiente para un perceptrón multicapa para calcular una aproximación uniforme ε a un conjunto de entrenamiento dado representado por el conjunto de entradas $x_1, ..., x_p$ y una salida deseada $f(x_1, ..., x_p)$. Sin embargo, el teorema no dice que una única capa oculta sea lo óptimo en el sentido de tiempo de aprendizaje o facilidad de implementación.

4.4.2 Consideraciones prácticas

El teorema de la aproximación universal es importante desde el punto de vista teórico porque proporciona la herramienta matemática necesaria para la viabilidad de redes feedforward con una capa oculta como una clase de soluciones de aproximación. Sin un teorema de este tipo, podríamos estar buscando una solución que no existe. Sin embargo, el teorema es una prueba de existencia; no nos dice cómo construir el perceptrón multicapa para realizar la aproximación.

Este teorema tiene un limitado valor práctico. El teorema asume que la función continua que pretende aproximarse es conocida y que se dispone de una capa oculta de tamaño ilimitado para la aproximación. Ambos supuestos no se cumplen en la mayoría de aplicaciones prácticas de los perceptrones multicapa.

El problema que presentan los perceptrones multicapa con una única capa oculta es que las neuronas tienden a interactuar unas con otras globalmente. En situaciones complejas, esta interacción hace que sea

difícil mejorar la aproximación en un punto sin empeorarla en otro punto. En cambio, con dos capas ocultas, el proceso de aproximación se hace más manejable. En particular, se puede procede como sigue:

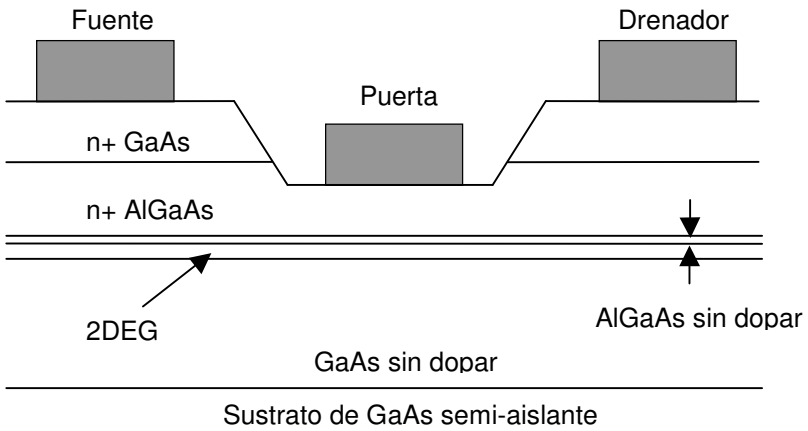
1. Características locales son extraídas en la primera capa oculta. Específicamente, algunas neuronas en la primera capa oculta son usadas para dividir el espacio de entrada en regiones, y otras neuronas en esta capa aprenden características locales de estas regiones.

2. Características globales son extraídas en la segunda capa oculta. Específicamente, una neurona de la segunda capa oculta combina las salidas de neuronas de la primera que están operando en una determinada región del espacio de entrada, y por tanto aprenden las características globales de esa región..

Este proceso de aproximación en dos etapas es similar en filosofía a la técnica de spline empleada para el ajuste de curvas, en el sentido de que los efectos de las neuronas están aislados y las aproximaciones en regiones diferentes del espacio de entrada pueden ser ajustadas individualmente.

5 APLICACIÓN DE LAS REDES NEURONALES PARA LA APROXIMACIÓN DE FUNCIONES CARACTERÍSTICAS DE UN TRANSISTOR HEMT

Los HEMT (high electron mobility transistor) son transistores de efecto de campo fabricados en un material de heterojunción. Existen varias maneras de realizar los HEMT. En la siguiente figura se muestra una de tales estructuras posibles:



La parte central de este dispositivo es la interface longitudinal entre una capa de GaAs no dopado y una capa de AlGaAs dopado con silicio. Estas capas son muy delgadas; la capa espaciadora entre ambas es apenas de 50 Armstrongs y deben ser fabricadas con técnicas epitaxiales como MBE o MOCVD.

El HEMT opera de un modo semejante al de un MOSFET. El canal es creado en la interface donde el material de AlGaAs le entrega electrones a la capa de GaAs. Estos electrones son mantenidos en una delgada capa cerca de la superficie. La densidad de electrones en esta delgada capa (llamada gas de electrones bidimensional) es controlada por el voltaje aplicado a la puerta; haciendo la tensión de puerta más positiva se incrementa la densidad de electrones y por lo tanto la corriente del canal también aumenta.

Debido a que no hay iones de impurezas en el GaAs que pudieran dispersar a los electrones, ellos se mueven a una alta velocidad a través del material. Como consecuencia, la velocidad de saturación de los electrones y su movilidad son mucho mas grandes en el HEMT que en el MESFET: la movilidad a campo eléctrico débil y a temperatura ambiente en el HEMT es de 8000 mientras que en el MESFET es de 5000. Aparte de esto, al reducirse la temperatura, la movilidad de los electrones decrece significativamente. Por esta razón, los HEMT a veces son operados a bajas temperaturas reduciéndose con esto la figura de ruido del dispositivo.

En electrónica, resulta muchas veces complicado hallar una expresión matemática que modele el comportamiento de un determinado dispositivo a partir de mediciones realizadas sobre éste. Normalmente se emplean modelos muy complicados basados en estudios profundos acerca del comportamiento de los semiconductores. Como ya vimos con anterioridad, una de las utilidades de las redes neuronales es la aproximación de funciones. Para el objetivo de modelar un dispositivo, las redes neuronales nos van a servir de gran utilidad, ya que nos permitirán encontrar una expresión matemática que aproxime el comportamiento de éste sin necesidad de estudio teórico, simplemente a partir de los datos obtenidos de las mediciones.

En este capítulo vamos a emplear las redes neuronales para, a partir de los datos experimentales obtenidos mediante mediciones de tensiones, intensidades y transconductancias de un par de transistores, obtener una expresión matemática que aproxime las curvas características del comportamiento de dichos transistores. Para ello emplearemos dos transistores, cada uno con un tamaño diferente.

También existen en la literatura diversos modelos matemáticos que intentan modelar el comportamiento de estos transistores. Nosotros emplearemos las redes neuronales para determinar los parámetros que definen dichos modelos.

5.1 Aproximación de la curva I_{ds} frente a V_{gs}

En este apartado vamos a partir de los datos obtenidos experimentalmente acerca de la intensidad de drenador en función de la tensión de puerta aplicada para intentar modelar la función característica de éstos mediante una red neuronal, lo que nos proporcionará una expresión matemática sencilla. Emplearemos una red neuronal con una única capa oculta, empleando el algoritmo Backpropagation. Además, realizaremos los análisis con una, dos y tres neuronas. Evidentemente, a mayor número de neuronas, en principio, la aproximación de la curva será mejor, pero consecuentemente la expresión matemática que describa dicha función será más compleja.

5.1.1 Transistor HEMT modelo foundry ED02AH con W=2x40 y V_{ds} = 3 V

A continuación se muestra una tabla en la que se reflejan los valores de Ids frente a Vgs para este transistor obtenidos experimentalmente y que nos servirán para entrenar nuestra red neuronal:

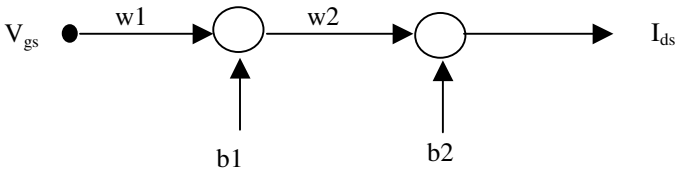
Vgs	Id	Vgs	Id
-2.2	0	-0.8	0.7597
-2.1	0	-0.7	1.5531
-2	0	-0.6	2.8469
-1.9	0.0001	-0.5	4.7377
-1.8	0.0002	-0.4	8.243
-1.7	0.0005	-0.3	12.94
-1.6	0.0011	-0.2	18.1968
-1.5	0.0025	-0.1	23.5825
-1.4	0.0057	0	28.7336
-1.3	0.013	0.1	33.3579
-1.2	0.0299	0.2	37.2825
-1.1	0.0684	0.3	40.4852
-1	0.1557	0.4	42.8287
-0.9	0.3492	0.5	43.2638

La función de transferencia que emplearemos será la tangente hiperbólica, ya que su forma se aproxima a la figura que forman los datos obtenidos experimentalmente. A continuación se muestran los distintos casos estudiados.

5.1.1.1 Empleando una neurona

A la hora de representar las redes neuronales, realizaremos una representación en forma de grafos, donde cada neurona vendrá representada por un círculo blanco y estarán conectadas mediante flechas que representarán los enlaces sinápticos.

En este caso, nuestra red neuronal tendrá el siguiente esquema:



Esta red neuronal consta de un nodo a la entrada, otro a la salida que simplemente es una suma ponderada, y una etapa intermedia con una neurona, cuya función de transferencia será una tangente hiperbólica, ya que si representamos los valores de I_{ds} frente a V_{gs} de este transistor siguen una curva muy parecida a la tangente hiperbólica.

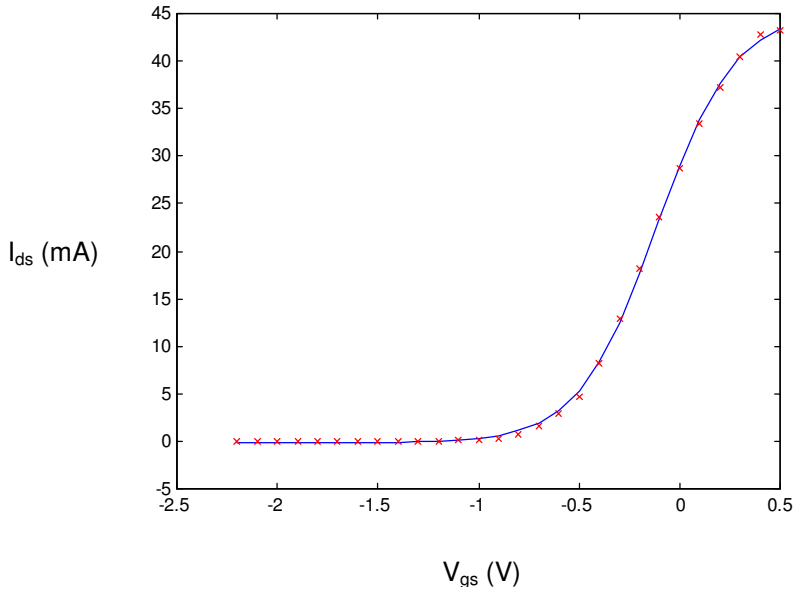
Mediante esta red neuronal se obtiene un buen resultado, con un error cuadrático medio de 0.0977493, con los siguientes valores para los parámetros indicados en el esquema anterior:

w1	**b1**	**w2**	**b2**
2.5830	0.2962	22.6220	22.4988

La expresión matemática que está realizando esta red neuronal es la siguiente:

$$I_{ds} = 22.4988 + 22.6220 \cdot tanh\left(2.5830 \cdot V_{gs} + 0.2962\right)$$

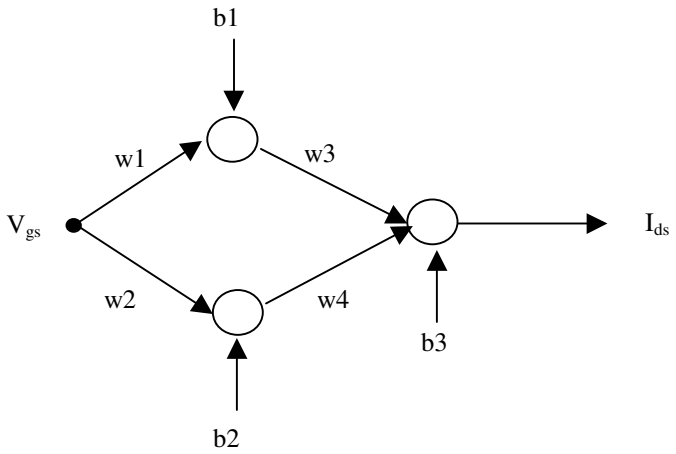
En la siguiente gráfica se encuentran representados tanto los datos utilizados para entrenar la red neuronal (cruces rojas) como la salida de la red neuronal para distintos valores de Vgs:



En esta gráfica se puede ver cómo la red neuronal, con tan solo una neurona, es capaz de aproximar bastante bien la función de transferencia del transistor.

5.1.1.2 Empleando dos neuronas

Para el caso de dos neuronas tenemos el siguiente esquema:



En este caso tenemos dos neuronas en la capa intermedia con función de transferencia tangente hiperbólica y el nodo de salida realiza una suma ponderada de las salidas de las dos neuronas de la capa intermedia.

Al simular esta red se obtienen los siguientes valores para los parámetros:

w1	**w2**	**w3**	**w4**
0.11878	-2.583	-11.751	-22.662

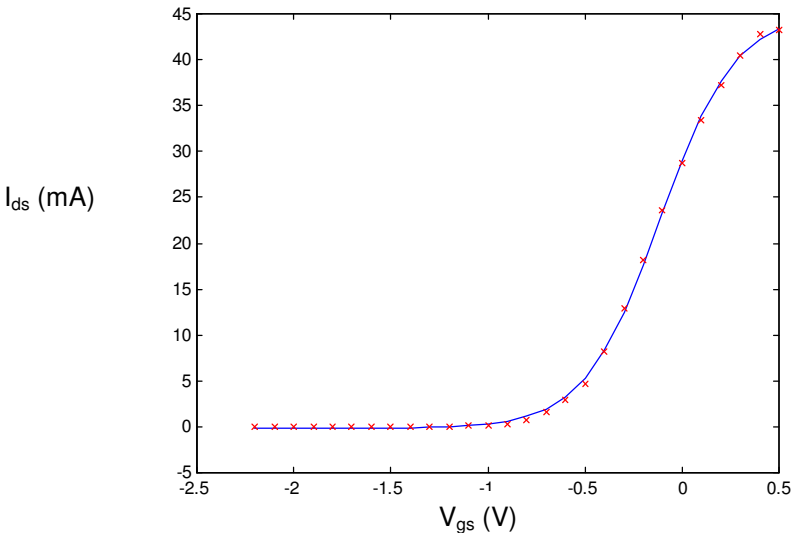
b1	**b2**	**b3**
-19.529	-0.2962	10.7474

El error cuadrático medio obtenido es el mismo que para el caso de una neurona. Vamos a ver la expresión matemática que está realizando esta red neuronal para ver qué está pasando:

$$I_{ds} = 10.7474 - 11.751 \cdot tanh(0.11878 \cdot V_{gs} - 19.529) - 22.662 \cdot tanh(-2.583 \cdot V_{gs} - 0.2962)$$

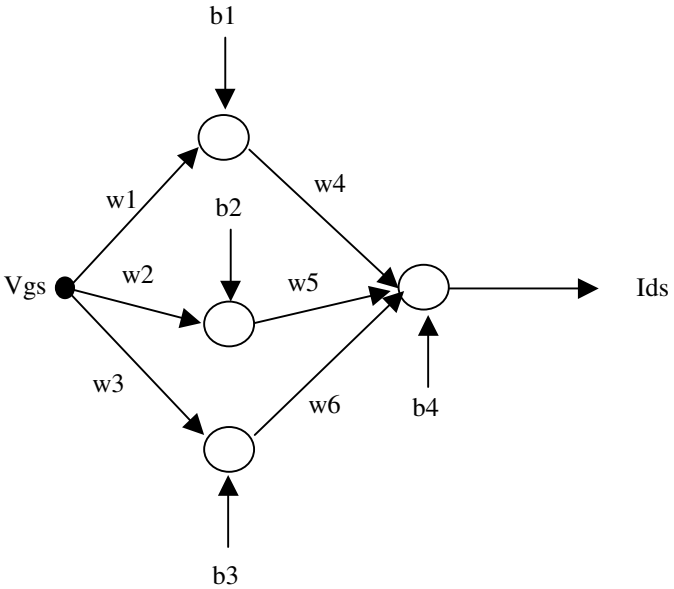
Si se observa con detenimiento la expresión, se puede ver que una de las dos neuronas no está haciendo otra cosa que sumar un término constante, ya que para el rango de valores de V_{gs} empleado, el término tanh (0.11878 · Vgs - 19.529) vale siempre –1. Por lo tanto, el introducir una neurona más no ha mejorado en nada la aproximación. Esto es debido a que los valores obtenidos experimentalmente del transistor sigue una curva prácticamente similar a una tangente hiperbólica, de ahí que el añadir otra tangente hiperbólica no implique mejoría alguna en la aproximación.

En la siguiente gráfica se puede observar la exactitud de la aproximación:



5.1.1.3 Empleando tres neuronas

En este caso tenemos el siguiente esquema:



Ahora hay tres neuronas en la capa intermedia con función de transferencia tangente hiperbólica.

Al entrenar esta red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

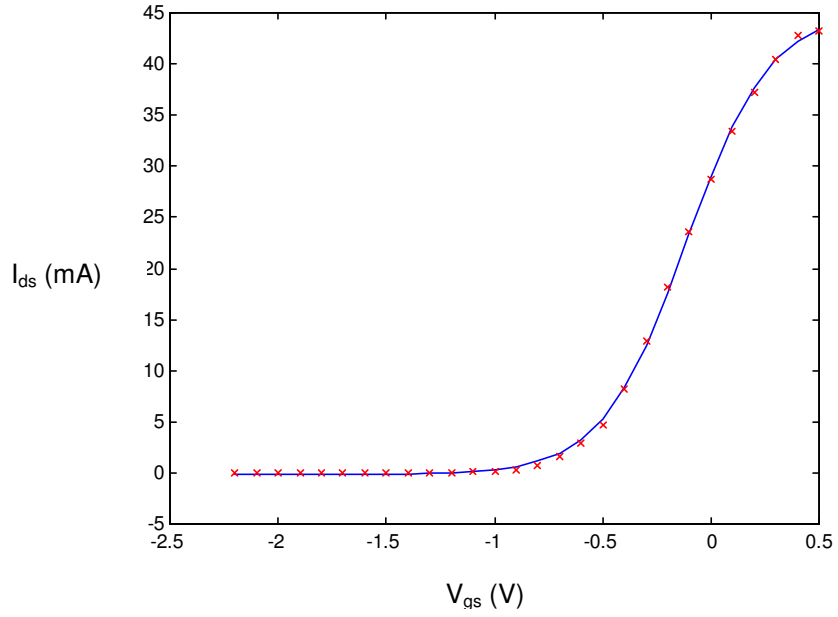
w1	w2	w3	w4	w5	W6
-0.162	0.0151	2.5830	-7.8275	-5.8391	22.6620

b1	b2	b3	b4
-19.699	-19.538	0.2962	8.8321

En este caso se obtiene un error cuadrático medio similar a los dos casos anteriores. La expresión matemática que está realizando la red neuronal es la siguiente:

$$I_{ds} = 8.8321 - 7.8275 \cdot tanh(-0.162 \cdot V_{gs} - 19.699) - 5.8391 \cdot tanh(0.0151 \cdot V_{gs} - 19.538) + 22.6620 \cdot tanh(2.5830 \cdot V_{gs} + 0.2962)$$

En este caso también se puede observar que tan solo está trabajando una neurona; las otras dos neuronas simplemente suman un término constante. En la siguiente gráfica se puede observar la exactitud de la aproximación:



Por lo tanto, se puede concluir que para el caso que nos ocupa de aproximación de la curva I_{ds} frente a V_{gs} para un transistor HEMT modelo foundry ED02AH con W=2x40 y Vds = 3 V., con una neurona en la capa intermedia con función de transferencia hiperbólica se puede conseguir una buena aproximación y obteniéndose una expresión matemática bastante sencilla.

5.1.2 Transistor HEMT modelo foundry ED02AH con W=6x50 y V_{ds} = 1.4 V

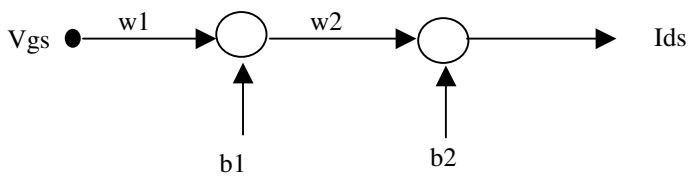
A continuación se muestra una tabla en la que se reflejan los valores de I_{ds} frente a V_{gs} para este transistor obtenidos experimentalmente y que nos servirán para entrenar nuestra red neuronal:

Vgs	Id	Vgs	Id
-2.2	0	-0.8	1.4618
-2.1	0	-0.7	2.9891
-2	0.0001	-0.6	5.485
-1.9	0.0002	-0.5	9.6412
-1.8	0.0004	-0.4	20.1403
-1.7	0.0009	-0.3	35.9784
-1.6	0.0021	-0.2	54.7233
-1.5	0.0048	-0.1	74.4507
-1.4	0.0109	0	93.553
-1.3	0.0251	0.1	110.7993
-1.2	0.0575	0.2	125.4748
-1.1	0.1317	0.3	137.4664
-1	0.2996	0.4	146.2467
-0.9	0.672	0.5	147.8751

A continuación se muestran los distintos casos estudiados.

5.1.2.1 Empleando una neurona

En este caso, nuestra red neuronal tendrá el siguiente esquema:



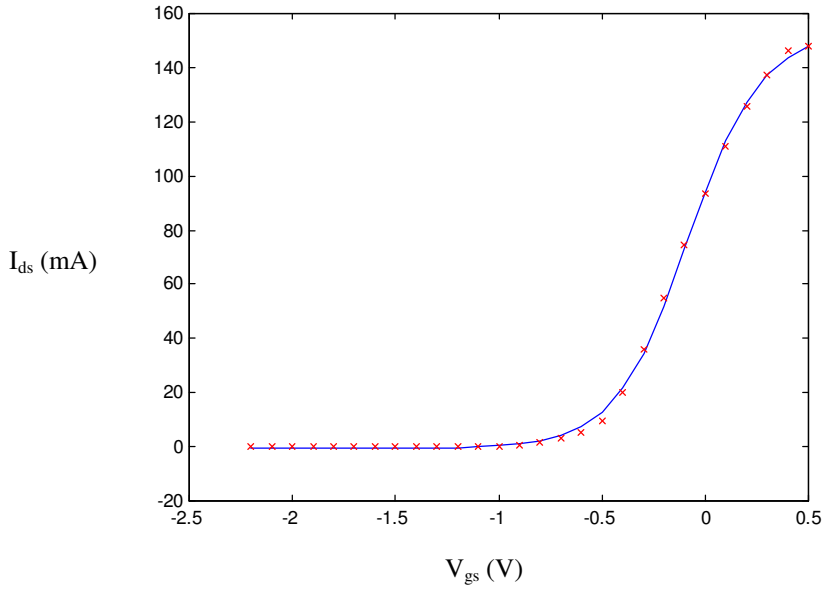
Los valores obtenidos se muestran a continuación:

w1	b1	w2	b2
2.8210	0.2352	76.9299	76.4131

El error cuadrático medio toma un valor de 1.69802. La expresión matemática que realiza la red neuronal es la siguiente:

$$I_{ds} = 76.413 + 76.9299 \cdot tanh\left(2.8210 \cdot V_{gs} + 0.2352\right)$$

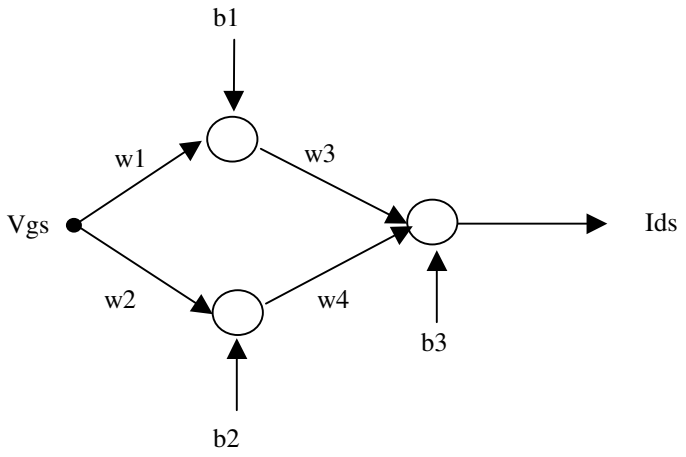
En la siguiente gráfica se puede apreciar la bondad de la aproximación:



Se puede ver que esta red neuronal tan simple es capaz de aproximar muy bien el comportamiento del transistor al igual que ocurrió en el caso del otro modelo.

5.1.2.2 Empleando dos neuronas

Para el caso de dos neuronas, como ya vimos, tenemos el siguiente esquema:



Los valores obtenidos después de entrenar la red neuronal son los siguientes:

w1	w2	w3	w4
0.5760	-2.8210	38.8808	-76.9299

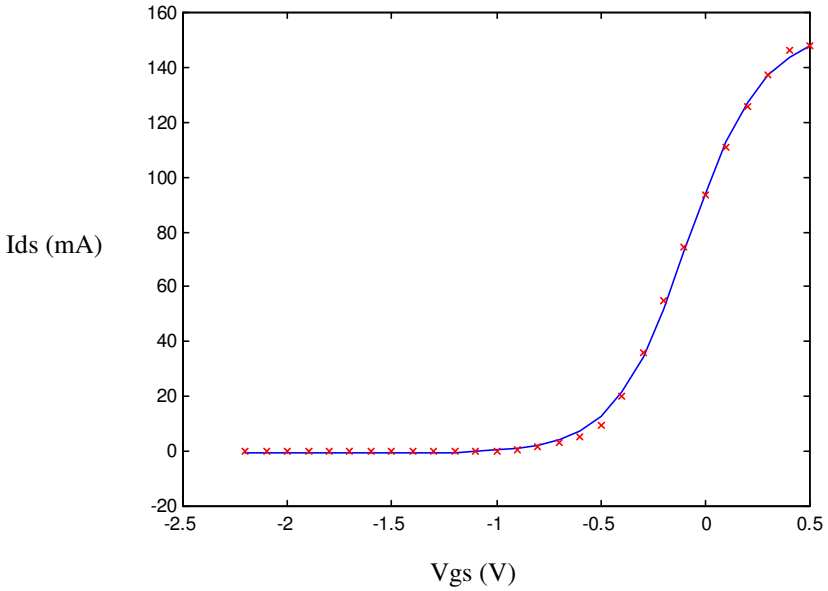
b1	b2	b3
21.4656	-0.2352	37.5323

El error cuadrático medio obtenido es el mismo que para el caso de una neurona. Vamos a ver la expresión matemática que está realizando esta red neuronal:

$$I_{ds} = 37.5323 + 38.8808 \cdot tanh(0.576 \cdot V_{gs} + 21.4656) - 76.9299 \cdot tanh(-2.821 \cdot V_{gs} - 0.2352)$$

Si se observa con detenimiento la expresión, se puede ver que de nuevo una de las dos neuronas no está haciendo otra cosa que sumar un término constante, ya que para el rango de valores de Vgs empleado el término tanh (0.5760 · Vgs + 21.4656) vale siempre 1. Por lo tanto, el introducir una neurona más no ha mejorado en nada la aproximación. Esto es debido , al igual que ocurría con el otro modelo de transistor, a que los valores obtenidos experimentalmente del transistor sigue una curva prácticamente similar a una tangente hiperbólica, de ahí que el añadir otra tangente hiperbólica no implique mejoría alguna en la aproximación.

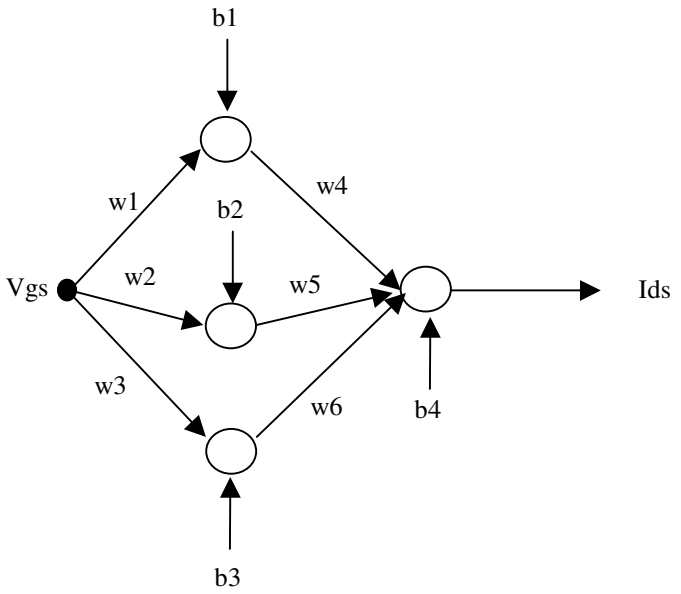
En la siguiente gráfica se puede observar la exactitud de la aproximación:



Se puede ver que el resultado es similar al obtenido con una neurona.

5.1.2.3 Empleando tres neuronas

Finalmente, para el caso de tres neuronas, tenemos el siguiente esquema:



Al entrenar esta red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

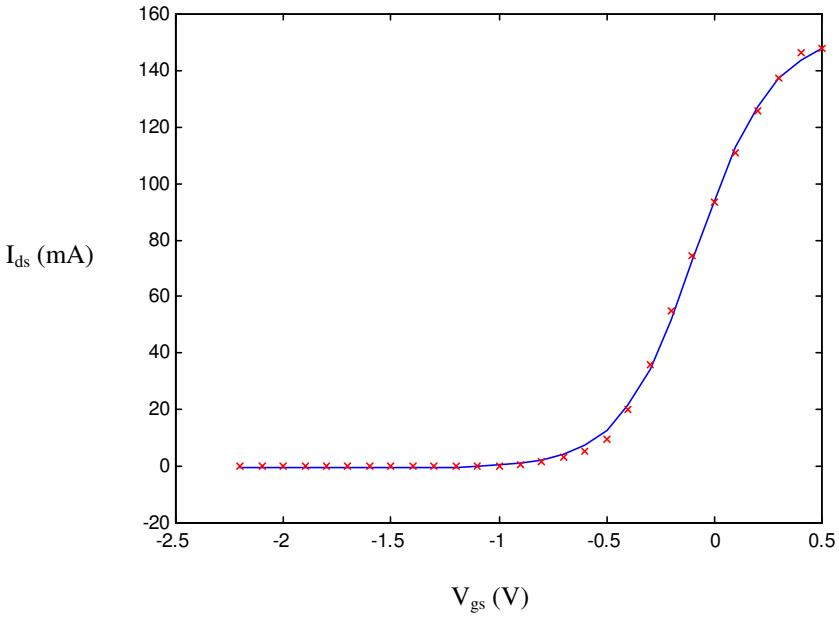
w1	**w2**	**w3**	**w4**	**w5**	**W6**
-0.1818	-0.2389	2.8210	29.3048	16.8543	76.9299

b1	**b2**	**b3**	**b4**
18.8570	17.9146	0.2352	30.2540

En este caso se obtiene un error cuadrático medio similar a los dos casos anteriores. La expresión matemática que está realizando la red neuronal es la siguiente:

$$I_{ds} = 30.254 + 29.3048 \cdot tanh(-0.1818 \cdot V_{gs} + 18.857) + 16.8543 \cdot tanh(-0.2389 \cdot V_{gs} + 17.9146) + \\ + 76.9299 \cdot tanh(2.821 \cdot V_{gs} + 0.2352)$$

En este caso también se puede observar que tan solo está trabajando una neurona; las otras dos neuronas simplemente suman un término constante. En la siguiente gráfica se puede observar la exactitud de la aproximación:



Por lo tanto se puede concluir también para este caso que la aproximación de la curva I_{ds} frente a V_{gs} para un transistor HEMT modelo foundry ED02AH con W=6x50 y V_{ds} = 1.4 V., con una neurona en la capa intermedia con función de transferencia hiperbólica se puede conseguir una buena aproximación y obteniéndose una expresión matemática bastante sencilla.

5.2 Aproximación de la curva g_m frente a V_{gs}

En esta apartado se va a tratar de ajustar la curva de la transconductancia del transistor en función de la tensión puerta-fuente, en lugar de la intensidad de drenador. En este caso, debido a la forma que presenta esta curva, utilizaremos una función exponencial cuadrática en lugar de la tangente hiperbólica como función de transferencia de las neuronas de la red neuronal, y realizaremos de nuevo el análisis con una, dos y tres neuronas. También vamos a estudiar el resultado obtenido con la función tangente hiperbólica.

5.2.1 Transistor HEMT modelo foundry ED02AH con W=2x40 y V_{ds} = 3 V

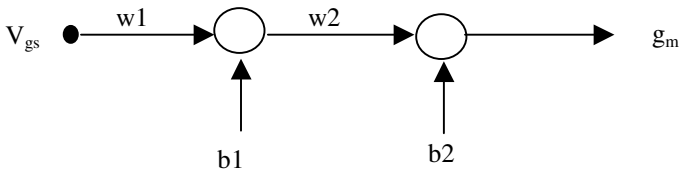
A continuación se muestra una tabla en la que se reflejan los valores de la transconductancia del transistor frente a V_{gs} obtenidos experimentalmente y que nos servirán para entrenar nuestra red neuronal:

V_{gs}	g_m	V_{gs}	g_m
-2.00000000000000	0.00000032159103	-0.70000000000000	0.01039305927445
-1.95000000000000	0.00000048651697	-0.65000000000000	0.01299681528846
-1.90000000000000	0.00000073651290	-0.60000000000000	0.01525662046213
-1.85000000000000	0.00000111551781	-0.55000000000000	0.01777546347557
-1.80000000000000	0.00000169017165	-0.50000000000000	0.02619008338242
-1.75000000000000	0.00000256154111	-0.45000000000000	0.03545992705825
-1.70000000000000	0.00000388289722	-0.40000000000000	0.04234552552023
-1.65000000000000	0.00000588666107	-0.35000000000000	0.04722153266871
-1.60000000000000	0.00000892522956	-0.30000000000000	0.05058139102237
-1.55000000000000	0.00001353279372	-0.25000000000000	0.05274161098699
-1.50000000000000	0.00002051885815	-0.20000000000000	0.05385977213151
-1.45000000000000	0.00003110952071	-0.15000000000000	0.05400960560160
-1.40000000000000	0.00004716044870	-0.10000000000000	0.05324468181559
-1.35000000000000	0.00007147689041	-0.05000000000000	0.05163479704162
-1.30000000000000	0.00010829211094	0	0.04928089301252
-1.25000000000000	0.00016397715972	0.05000000000000	0.04631786637777
-1.20000000000000	0.00024808114788	0.10000000000000	0.04291169570824
-1.15000000000000	0.00037482687587	0.15000000000000	0.03925351318134
-1.10000000000000	0.00056519453589	0.20000000000000	0.03554888238874
-1.05000000000000	0.00084967214780	0.25000000000000	0.03198932088903
-1.00000000000000	0.00127153530551	0.30000000000000	0.02863323436917
-0.95000000000000	0.00188993986564	0.35000000000000	0.02461687924594
-0.90000000000000	0.00278083081776	0.40000000000000	0.01338932491216
-0.85000000000000	0.00403130369920	0.45000000000000	0.00297064691907
-0.80000000000000	0.00571974860951	0.50000000000000	0.00097319226241
-0.75000000000000	0.00787226368081		

Una vez entrenada la red, comprobaremos la salida de la red neuronal para los distintos valores de la tensión de puerta y comprobaremos los resultados con los datos experimentales reflejados en la tabla anterior.

5.2.1.1 Empleando una neurona

El esquema que utilizaremos será el siguiente:



Este esquema es el mismo que el empleado en el apartado anterior para el caso de una neurona. Analizaremos el resultado obtenido cuando la función de transferencia de la neurona sea una exponencial, que se ajusta mejor a la curva que describe g_m frente a V_{gs}. En este caso, con tan solo una neurona, no tendrá sentido hacer el análisis con la función tangente hiperbólica, ya que nunca podría tener una forma parecida. Más adelante, en el caso de dos y tres neuronas, sí analizaremos los resultados que se pueden obtener tanto con la función exponencial como con la tangente hiperbólica.

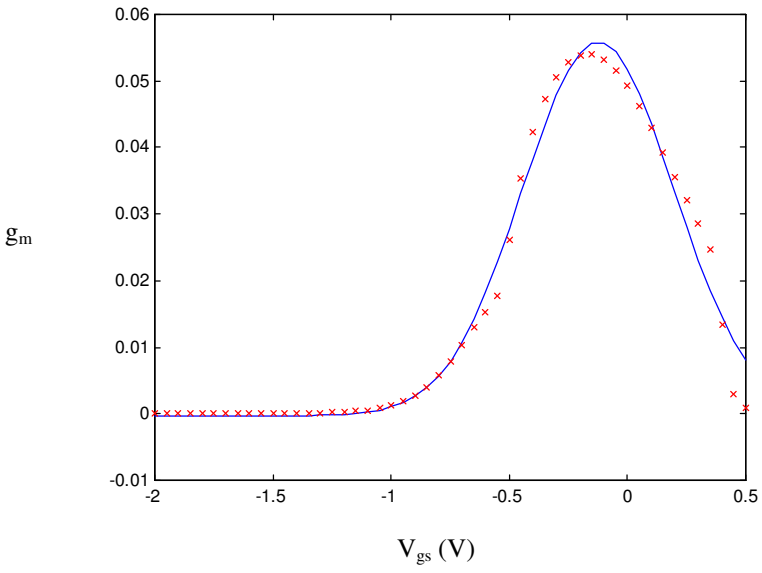
Tras entrenar la red, se obtuvo un error cuadrático medio de $6.20138 \cdot 10^{-6}$, con los siguientes valores para los distintos parámetros de la red:

w1	**b1**	**w2**	**b2**
2.2112	0.2731	0.0561	$-2.9361 \cdot 10^{-4}$

La expresión matemática que está realizando esta red neuronal es la siguiente:

$$g_m = -2.9361 \cdot 10^{-4} + 0.0561 \cdot e^{-(2.2112 \cdot V_{gs} + 0.2731)^2}$$

Y en la siguiente gráfica se puede ver la exactitud de la representación:

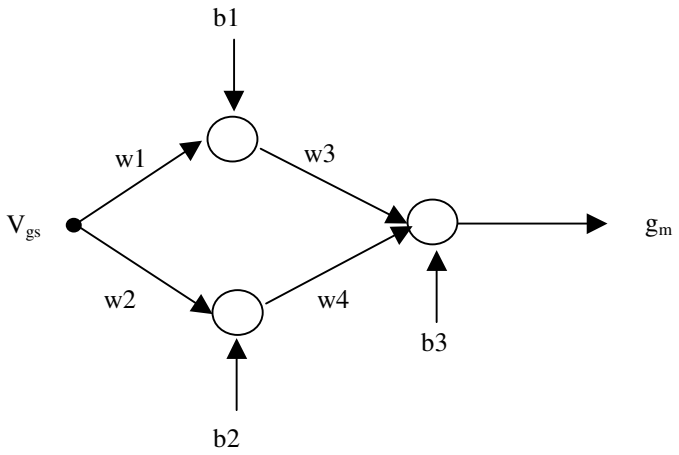


Como se puede ver, la red neuronal es capaz de seguir más o menos los valores de la transconductancia, desviándose con mayor notoriedad en los valores altos de V_{gs}.

Aquí se puede ver que si empleáramos como función de transferencia la tangente hiperbólica no se podría obtener un buen resultado debido a la distinta naturaleza de ambas curvas. Como con una neurona no se puede obtener un buen resultado, vamos a pasar a estudiar qué se puede conseguir con dos neuronas.

5.2.1.2 Empleando dos neuronas

Para el caso de dos neuronas tenemos el siguiente esquema:



Vamos a proceder a analizar este caso tanto para el caso de la función de transferencia exponencial como para el caso de la tangente hiperbólica.

5.2.1.2.1 Con función de activación exponencial cuadrática

En este caso tenemos dos neuronas en la capa intermedia con función de transferencia exponencial cuadrática.

Al simular esta red se obtienen los siguientes valores para los parámetros:

w1	**w2**	**w3**	**w4**
10.5691	-2.1228	-0.0101	0.0553

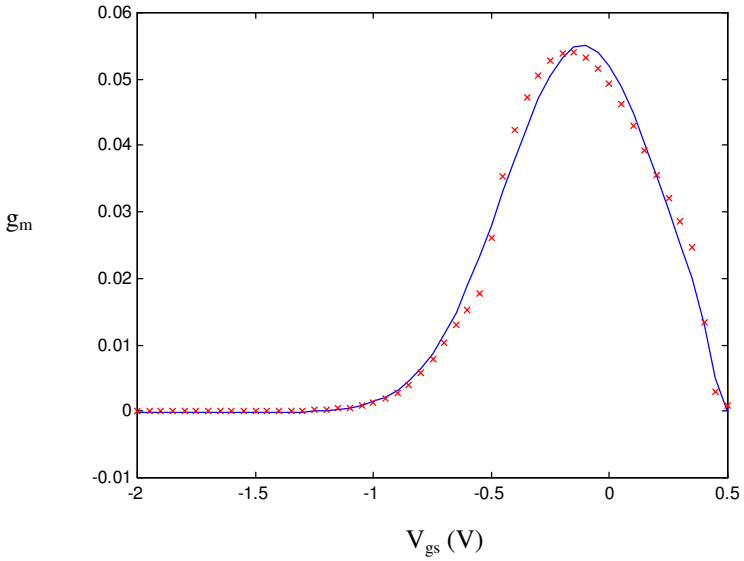
b1	b2	b3
-5.2425	-0.2410	$-1.8724\cdot10^{-4}$

El error cuadrático medio obtenido en esta ocasión es de $3.64444\cdot10^{-6}$, lo cual mejora el resultado obtenido para el caso de una neurona, como era de esperar.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = -1.8724\cdot10^{-4} - 0.0101\cdot e^{-(10.5691\cdot V_{gs} - 5.2425)^2} + 0.0553\cdot e^{-(-2.1228\cdot V_{gs} - 0.241)^2}$$

En la siguiente gráfica se puede comprobar como con dos neuronas se consigue una mejor aproximación de la transconductancia:



Vemos como al introducir una segunda neurona se mejora el comportamiento de la red para el caso de valores altos de Vgs, que es la parte más complicada de seguir.

5.2.1.2.2 Con función de activación tangente hiperbólica

En este apartado vamos a estudiar qué resultados se pueden conseguir con la función de transferencia hiperbólica a pesar de no tener forma similar a la que describe la transconductancia del transistor. Pero el hecho de que contemos ahora con dos neuronas introduce una mayor flexibilidad a la red neuronal para ajustarse mejor a dicha curva.

Al simular la red se obtienen los siguientes valores para los distintos parámetros:

w1	w2	w3	w4
2.0281	4.3480	-0.0736	0.0322

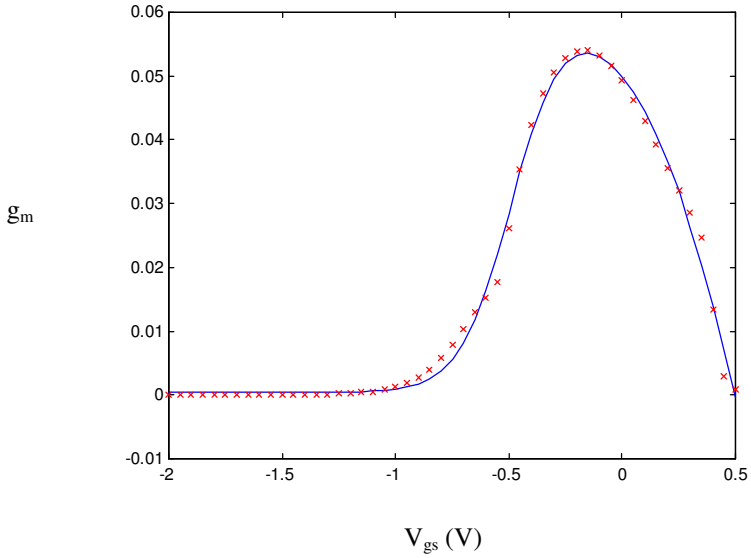
b1	b2	b3
-1.1305	2.1062	-0.0410

El error cuadrático medio obtenido en esta ocasión es de $2.00506 \cdot 10^{-6}$, lo cual mejora el resultado obtenido para el caso de la función de transferencia exponencial cuadrática.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = -0.041 - 0.0736 \cdot tanh\left(2.0281 \cdot V_{gs} - 1.1305\right) + 0.0322 \cdot tanh\left(4.348 \cdot V_{gs} + 2.1062\right)$$

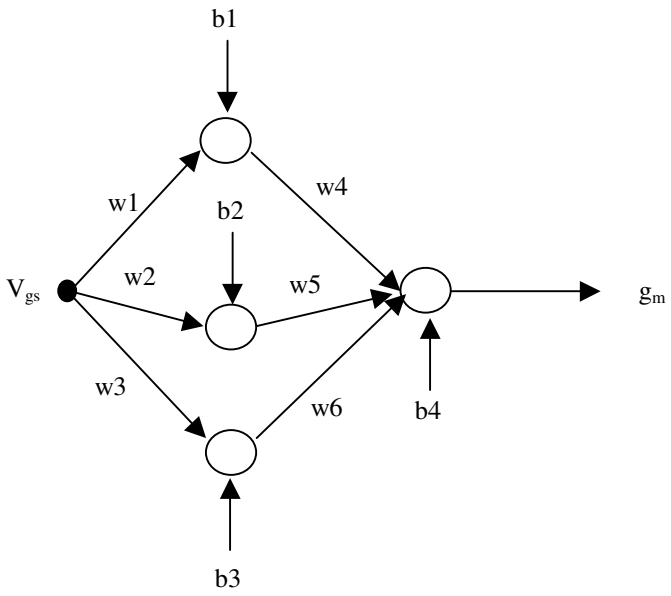
En la siguiente gráfica se puede comprobar la bondad de la aproximación:



Se puede ver cómo a pesar de que a la red le cuesta más trabajo seguir la subida inicial, es capaz de seguir mejor la curva para valores altos de V_{gs}.

5.2.1.3 Empleando tres neuronas

El esquema de la red neuronal es el siguiente:



A continuación vamos a ver los resultados obtenidos para las dos funciones de transferencia.

5.2.1.3.1 Con función de activación exponencial cuadrática

Al entrenar esta red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

w1	**w2**	**w3**	**w4**	**w5**	**w6**
-37.7387	4.2453	-2.6791	-0.0181	0.0191	0.0540

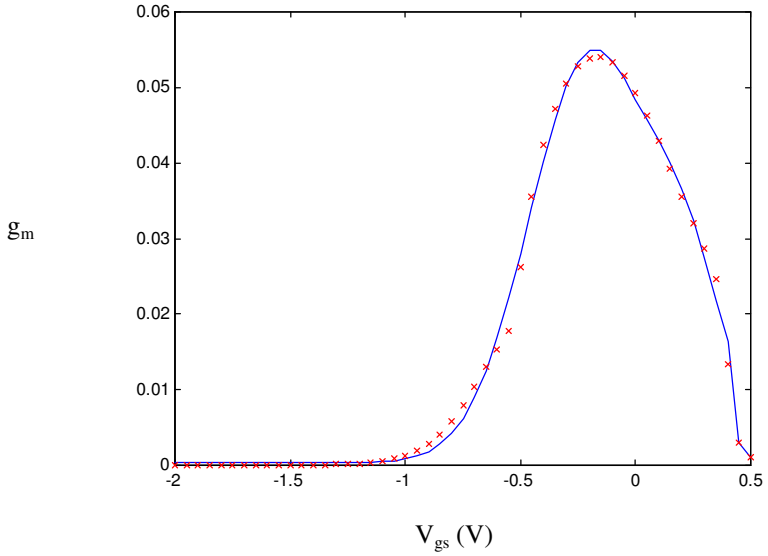
b1	**b2**	**b3**	**b4**
17.8550	-1.0016	-0.5209	$3.1461 \cdot 10^{-4}$

En este caso se obtiene un error cuadrático medio de $1.34746 \cdot 10^{-6}$, valor que mejora aun más los resultados anteriores.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = 3.1461 \cdot 10^{-4} - 0.0181 \cdot e^{-\left(-37.7387 \cdot V_{gs} + 17.8550\right)^2} + 0.0191 \cdot e^{-\left(4.2453 \cdot V_{gs} - 1.0016\right)^2} + \\ + 0.054 \cdot e^{-\left(-2.6791 \cdot V_{gs} - 0.5209\right)^2}$$

A continuación se muestra cómo la curva se aproxima aún mejor a los datos experimentales y es capaz de seguir mejor la curva para valores altos de V_{gs}:



5.2.1.3.2 Con función de transferencia tangente hiperbólica

Como ya vimos para el caso de dos neuronas, a pesar que la función tangente hiperbólica no tiene la misma forma que la de la transconductancia, si sumamos varias tangentes hiperbólicas sí que podremos obtener un resultado parecido. Vamos a ver qué pasa si añadimos una neurona más.

Al entrenar la red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

w1	**w2**	**w3**	**w4**	**w5**	**w6**
1.9582	4.2407	0	-0.0748	0.0329	0

b1	**b2**	**b3**	**b4**
-1.093	2.0416	0	-0.0414

En este caso se obtiene prácticamente el mismo error cuadrático medio que cuando teníamos dos neuronas. De los resultados obtenidos se puede ver que hay una neurona que no está trabajando, de ahí que no haya mejora. El añadir una tercera neurona no ha implicado mejora alguna en la red neuronal a la hora de aproximar la curva de la transconductancia del transistor.

5.2.2 Transistor HEMT modelo foundry ED02AH con W=6x50 y V_{ds} = 1.4 V

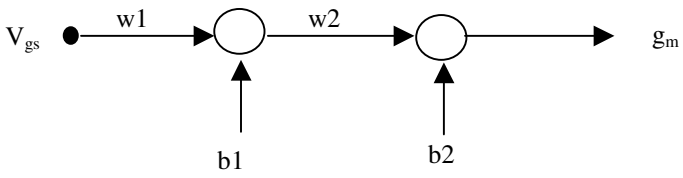
A continuación se muestra una tabla en la que se reflejan los valores de la transconductancia del transistor frente a V_{gs} obtenidos experimentalmente y que nos servirán para entrenar nuestra red neuronal:

V_{gs}	g_m	V_{gs}	g_m
-2.00000000000000	0.00000062519489	-0.70000000000000	0.02001001051987
-1.95000000000000	0.00000094347271	-0.65000000000000	0.02505061124189
-1.90000000000000	0.00000142559911	-0.60000000000000	0.02958406426479
-1.85000000000000	0.00000215614922	-0.55000000000000	0.03690144613079
-1.80000000000000	0.00000326338314	-0.50000000000000	0.06879599767560
-1.75000000000000	0.00000494180856	-0.45000000000000	0.10625641154513
-1.70000000000000	0.00000748638743	-0.40000000000000	0.13632927927675
-1.65000000000000	0.00001134438287	-0.35000000000000	0.15937904280058
-1.60000000000000	0.00001719391218	-0.30000000000000	0.17643174906162
-1.55000000000000	0.00002606289105	-0.25000000000000	0.18824100217275
-1.50000000000000	0.00003950897286	-0.20000000000000	0.19529332638568
-1.45000000000000	0.00005989138222	-0.15000000000000	0.19794810590144
-1.40000000000000	0.00009078069882	-0.10000000000000	0.19655899031139
-1.35000000000000	0.00013757458871	-0.05000000000000	0.19154338256731
-1.30000000000000	0.00020841836601	0	0.18341060938039
-1.25000000000000	0.00031557068057	0.05000000000000	0.17276640126453
-1.20000000000000	0.00047740519630	0.10000000000000	0.16030538073229
-1.15000000000000	0.00072128855072	0.15000000000000	0.14679479148957
-1.10000000000000	0.00108759017174	0.20000000000000	0.13304008657814
-1.05000000000000	0.00163497577028	0.25000000000000	0.11978283585992
-1.00000000000000	0.00244672122155	0.30000000000000	0.10725772709029
-0.95000000000000	0.00363667124902	0.35000000000000	0.09223616357476
-0.90000000000000	0.00535100813819	0.40000000000000	0.05015910437096
-0.85000000000000	0.00775745453843	0.45000000000000	0.01110640081203
-0.80000000000000	0.01100721597340	0.50000000000000	0.00362736836354
-0.75000000000000	0.01515159514153		

Una vez entrenada la red, comprobaremos la salida de la red neuronal para los distintos valores de la tensión de puerta y comprobaremos los resultados con los datos experimentales reflejados en la tabla anterior.

5.2.2.1 Empleando una neurona

El esquema utilizado es el siguiente:



En este caso usaremos solo la función de transferencia exponencial, al igual que hicimos con el otro modelo de transistor, ya que con la función de transferencia tangente hiperbólica es imposible obtener una buena aproximación.

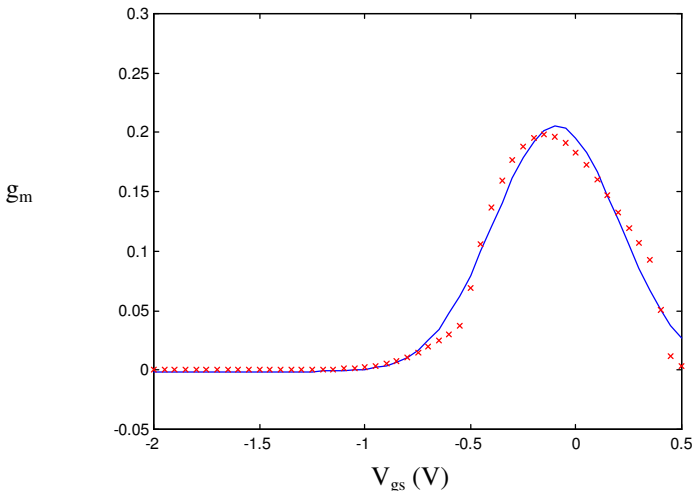
Tras entrenar la red neuronal, los valores obtenidos para los parámetros son los siguientes:

w1	**b1**	**w2**	**b2**
-2.3774	-0.2210	0.2070	-0.0014

Con estos parámetros se obtiene un error cuadrático medio de $1.05739 \cdot 10^{-4}$, teniendo la siguiente expresión matemática:

$$g_m = -0.0014 + 0.207 \cdot e^{-\left(-2.3774 \cdot V_{gs} - 0.221\right)^2}$$

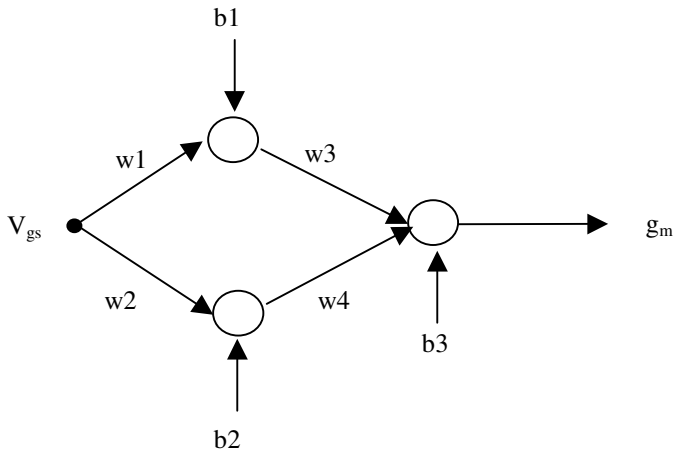
En la siguiente gráfica se puede comprobar la exactitud de la aproximación:



Se puede ver cómo le cuesta a la red neuronal seguir los valores experimentales, ya que no posee suficientes parámetros libres para implementar una expresión con dichas variaciones.

5.2.2.2 Empleando dos neuronas

Para el caso de dos neuronas volvemos a tener el siguiente esquema:



Para este caso volveremos a hacer el análisis tanto con la función exponencial como con la tangente hiperbólica, ya que ya vimos que al tener dos neuronas con la función tangente hiperbólica se podía conseguir una curva bastante parecida a la buscada.

5.2.2.2.1 Con función de transferencia exponencial cuadrática

Al simular esta red se obtienen los siguientes valores para los distintos parámetros:

w1	**w2**	**w3**	**w4**
5.1193	-3.0502	0.0867	0.2027

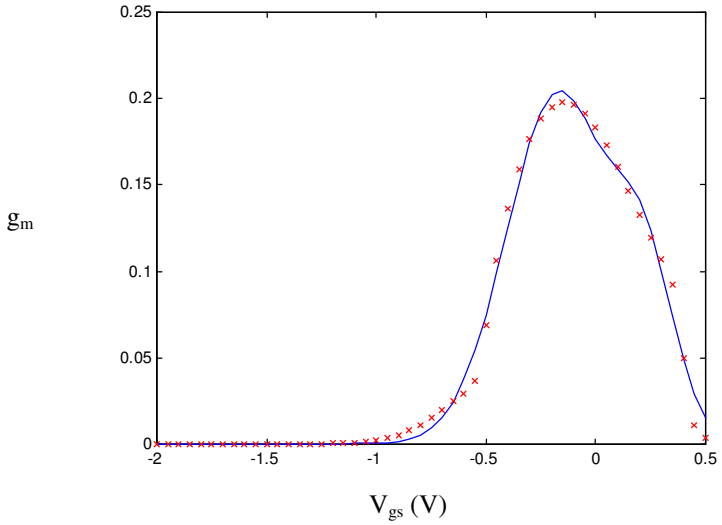
b1	**b2**	**b3**
-1.1556	-0.5255	$-3.0151\cdot10^{-4}$

El error cuadrático medio obtenido en esta ocasión es de $3.725\cdot10^{-5}$, lo cual mejora bastante el resultado obtenido para el caso de una neurona, como era de esperar.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = -3.0151\cdot10^{-4} + 0.0867\cdot e^{-(5.1193\cdot V_{gs} - 1.1556)^2} + 0.2027\cdot e^{-(-3.0502\cdot V_{gs} - 0.5255)^2}$$

En la siguiente gráfica se puede comprobar como con dos neuronas se consigue una mejor aproximación de la transconductancia:



5.2.2.2.2 Con función de transferencia tangente hiperbólica

Al simular la red se obtienen los siguientes valores para los distintos parámetros:

w1	**w2**	**w3**	**w4**
-2.5497	5.9167	0.2107	0.1079

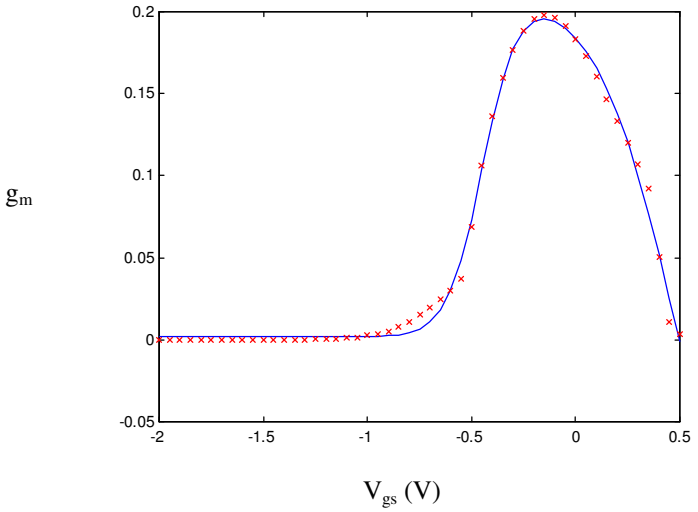
b1	**b2**	**b3**
1.2374	2.6336	-0.1010

El error cuadrático medio obtenido en esta ocasión es de $2.24555 \cdot 10^{-5}$, lo cual mejora el resultado obtenido para el caso de la función de transferencia exponencial.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = -0.101 + 2.107 \cdot tanh\left(-2.5497 \cdot V_{gs} + 1.2374\right) + 0.1079 \cdot tanh\left(5.9167 \cdot V_{gs} + 2.6336\right)$$

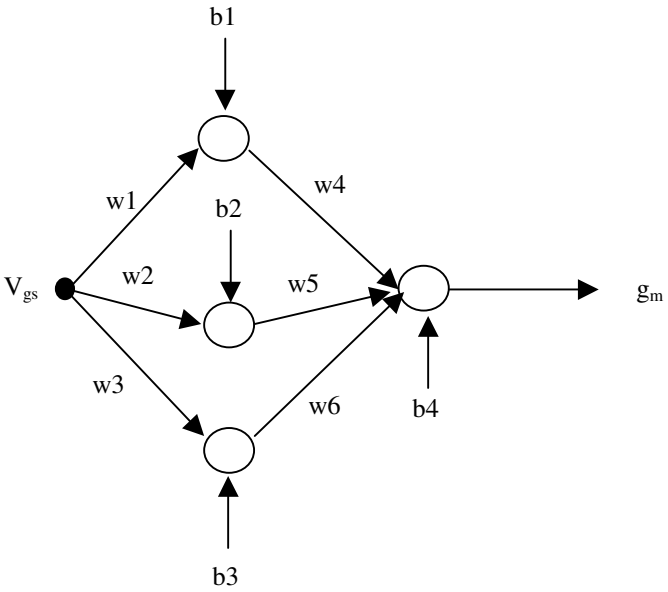
En la siguiente gráfica se puede comprobar la bondad de la aproximación:



Se ve cómo se obtiene una curva más suave que la conseguida con la función exponencial, además de tener un error cuadrático medio menor.

5.2.2.3 Empleando tres neuronas

El esquema de la red neuronal es el siguiente:



Vamos a ver los dos casos al igual que como hemos hecho hasta ahora.

5.2.2.3.1 Con función de transferencia exponencial cuadrática

Al entrenar esta red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

w1	w2	w3	w4	w5	w6
-37.7158	3.1929	-3.7048	-0.0783	0.1342	0.1579

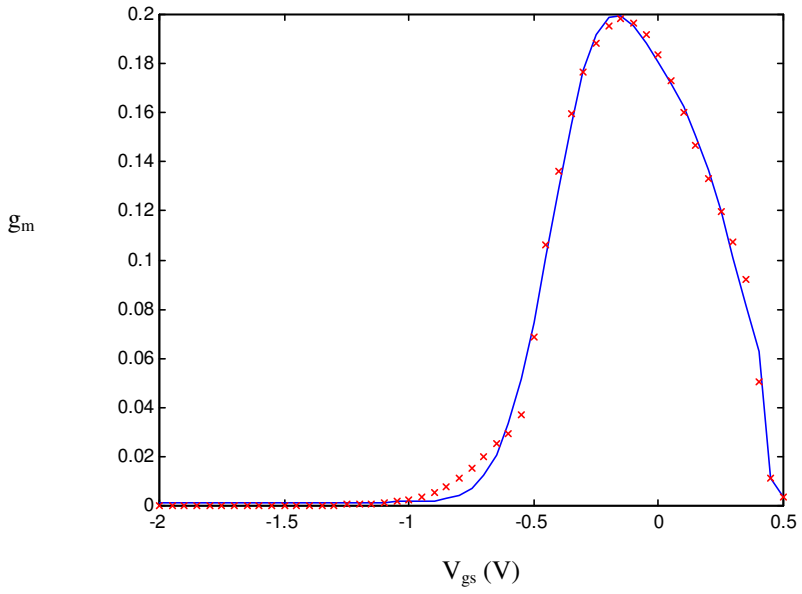
b1	b2	b3	b4
17.8638	-0.3897	-0.9523	0.0014

En este caso se obtiene un error cuadrático medio de $1.99862 \cdot 10^{-5}$, valor que mejora aun más los resultados anteriores.

La expresión matemática que está realizando la red neuronal es la siguiente:

$$g_m = 0.0014 - 0.0783 \cdot e^{-\left(-37.7158 \cdot V_{gs} + 17.8638\right)^2} + 0.1342 \cdot e^{-\left(3.1929 \cdot V_{gs} - 0.3897\right)^2} + \\ + 0.1579 \cdot e^{-\left(-3.7048 \cdot V_{gs} - 0.9523\right)^2}$$

A continuación se muestra cómo la curva se aproxima aún mejor a los datos experimentales y es capaz de seguir mejor la curva para valores altos de V_{gs}:



5.2.2.3.2 Con función de transferencia tangente hiperbólica

Vamos a ver qué pasa si, empleando como función de transferencia la función tangente hiperbólica, añadimos una neurona más, es decir, usamos tres neuronas en la capa oculta.

Al entrenar la red mediante los datos obtenidos experimentalmente se obtienen los siguientes valores para los distintos parámetros de la red:

w1	**w2**	**w3**	**w4**	**w5**	**w6**
-1.9126	-0.6166	-6.3845	0.4185	-0.1755	-0.1038

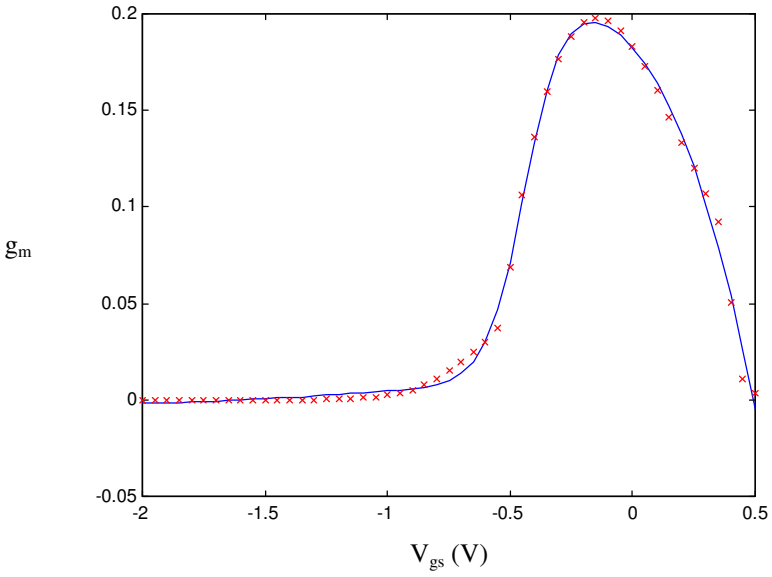
b1	**b2**	**b3**	**b4**
1.3381	1.1072	-2.7989	-0.1443

Dando lugar a la siguiente expresión matemática:

$$g_m = -0.1443 + 0.4185 \cdot tanh(-1.9126 \cdot V_{gs} + 1.3381) - 0.1755 \cdot tanh(-0.6166 \cdot V_{gs} + 1.1072) + \\ -0.1038 \cdot tanh(-6.3845 \cdot V_{gs} - 2.7989)$$

En este caso se obtiene una ligera mejora en el error cuadrático medio respecto a cuando teníamos dos neuronas.

En la siguiente gráfica se puede observar el resultado obtenido:



5.3 Aproximación de la curva I_{ds} frente a V_{gs}, V_{ds} (Modelo GSR1)

En este apartado vamos a estudiar un modelo matemático realizado por el Grupo de Sistema Radio de la Universidad de Sevilla mediante el cual se tiene la intensidad en función de la tensión de puerta y de la tensión de drenador. Vamos a crear una red neuronal que implemente dicha expresión matemática y vamos a determinar los valores de los parámetros presentándole a dicha red una batería de datos de entrada y salida. Vamos a utilizar de nuevo los datos correspondientes a los transistores HEMT modelo foundry ED02AH con W = 2x40 y con W = 6x50.

5.3.1 Transistor HEMT modelo foundry ED02AH con W=2x40

En las siguientes tablas se muestran los datos conocidos de la intensidad I_{ds} en función de las tensiones V_{gs} y V_{ds} y que utilizaremos para entrenar la red neuronal. En la primera fila se tienen los distintos valores de V_{ds}, mientras que en la primera columna se tienen los distintos valores de V_{gs} para los que se evalúa la intensidad de drenador.

	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
-2	0	0	0	0	0	0	0	0	0	0	0
-1.9	0	0	0	0	0	0	0	0	0	0	0
-1.8	0	0	0	0	0	0	0	0.0001	0.0001	0.0001	0.0001
-1.7	0	0	0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0002
-1.6	0	0	0.0001	0.0001	0.0002	0.0002	0.0003	0.0003	0.0003	0.0004	0.0004
-1.5	0	0.0001	0.0002	0.0003	0.0004	0.0005	0.0006	0.0007	0.0008	0.0008	0.0009
-1.4	0	0.0002	0.0005	0.0007	0.0009	0.0011	0.0013	0.0015	0.0017	0.0019	0.0021
-1.3	0	0.0005	0.001	0.0016	0.0021	0.0025	0.003	0.0035	0.004	0.0044	0.0049
-1.2	0	0.0012	0.0024	0.0036	0.0047	0.0058	0.0069	0.008	0.0091	0.0102	0.0113
-1.1	0	0.0028	0.0055	0.0081	0.0108	0.0133	0.0159	0.0184	0.0209	0.0233	0.0258
-1	0	0.0063	0.0124	0.0185	0.0245	0.0303	0.0361	0.0419	0.0475	0.0531	0.0586
-0.9	0	0.014	0.0278	0.0414	0.0548	0.068	0.0811	0.0939	0.1066	0.1191	0.1314
-0.8	0	0.0305	0.0606	0.0902	0.1193	0.148	0.1763	0.2043	0.2318	0.259	0.2859
-0.7	0	0.0625	0.124	0.1845	0.2441	0.3028	0.3607	0.4178	0.4741	0.5297	0.5846
-0.6	0	0.116	0.2296	0.3408	0.45	0.5576	0.6636	0.7682	0.8714	0.9731	1.0736
-0.5	0	0.3181	0.5803	0.7924	0.9779	1.1515	1.3194	1.4837	1.6454	1.8047	1.962
-0.4	0	1.3727	2.3097	2.8553	3.2027	3.4694	3.7047	3.927	4.1429	4.3547	4.5633
-0.3	0	3.2167	5.3106	6.396	6.9732	7.3459	7.6419	7.9084	8.1625	8.4102	8.6536
-0.2	0	5.5286	9.0627	10.8059	11.6477	12.1284	12.4749	12.7718	13.0489	13.3168	13.5794
-0.1	0	8.0244	13.1087	15.5533	16.67	17.2558	17.6455	17.9634	18.2539	18.5325	18.8046
0	0	10.4686	17.0691	20.1967	21.5781	22.2619	22.6889	23.0226	23.3214	23.6055	23.8823

0.1	0	12.6865	20.662	24.4079	26.0276	26.7985	27.2573	27.6034	27.9076	28.1949	28.4739
0.2	0	14.5782	23.7263	27.999	29.8212	30.6655	31.1507	31.5065	31.8147	32.1039	32.384
0.3	0	16.1258	26.233	30.9365	32.9241	33.8281	34.3345	34.698	35.0091	35.2995	35.5803
0.4	0	17.2596	28.0694	33.0883	35.1969	36.1446	36.6665	37.0355	37.3486	37.6398	37.921
0.5	0	17.4697	28.4096	33.487	35.6181	36.5739	37.0987	37.4687	37.7822	38.0736	38.3549

Valores de I_{ds} para valores de V_{gs} entre –2 y 0.5 y valores de V_{ds} entre 0 y 1

	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2**
-2	0	0	0	0	0	0	0	0	0	0
-1.9	0	0	0	0	0	0.0001	0.0001	0.0001	0.0001	0.0001
-1.8	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
-1.7	0.0002	0.0002	0.0002	0.0002	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
-1.6	0.0004	0.0005	0.0005	0.0006	0.0006	0.0006	0.0007	0.0007	0.0007	0.0008
-1.5	0.001	0.0011	0.0012	0.0013	0.0014	0.0014	0.0015	0.0016	0.0017	0.0017
-1.4	0.0023	0.0025	0.0027	0.0029	0.0031	0.0033	0.0035	0.0036	0.0038	0.004
-1.3	0.0054	0.0058	0.0063	0.0067	0.0071	0.0075	0.008	0.0084	0.0088	0.0092
-1.2	0.0123	0.0133	0.0143	0.0153	0.0163	0.0173	0.0183	0.0192	0.0202	0.0211
-1.1	0.0281	0.0305	0.0328	0.0351	0.0374	0.0396	0.0418	0.044	0.0462	0.0483
-1	0.064	0.0694	0.0747	0.0799	0.085	0.0901	0.0952	0.1002	0.1051	0.1099
-0.9	0.1436	0.1556	0.1675	0.1792	0.1908	0.2022	0.2135	0.2247	0.2357	0.2466
-0.8	0.3124	0.3385	0.3643	0.3898	0.415	0.4399	0.4645	0.4887	0.5127	0.5365
-0.7	0.6387	0.6922	0.745	0.7971	0.8486	0.8994	0.9497	0.9993	1.0484	1.0969
-0.6	1.1727	1.2706	1.3672	1.4627	1.5569	1.65	1.742	1.8329	1.9227	2.0115
-0.5	2.1172	2.2704	2.4216	2.571	2.7185	2.8643	3.0082	3.1505	3.2911	3.4301
-0.4	4.7691	4.9722	5.1727	5.3707	5.5663	5.7595	5.9504	6.139	6.3254	6.5096
-0.3	8.8934	9.13	9.3636	9.5942	9.8221	10.0471	10.2694	10.4891	10.7062	10.9208
-0.2	13.8378	14.0928	14.3444	14.5929	14.8383	15.0807	15.3202	15.5568	15.7907	16.0218
-0.1	19.0722	19.336	19.5964	19.8535	20.1074	20.3583	20.6061	20.8509	21.0929	21.3321
0	24.1541	24.422	24.6864	24.9475	25.2053	25.4599	25.7115	25.9601	26.2058	26.4487
0.1	28.7477	29.0175	29.2837	29.5465	29.806	30.0624	30.3157	30.5659	30.8133	31.0577
0.2	32.6588	32.9294	33.1964	33.4599	33.7202	33.9773	34.2314	34.4824	34.7304	34.9756
0.3	35.8555	36.1265	36.3938	36.6577	36.9183	37.1758	37.4301	37.6815	37.9298	38.1753
0.4	38.1964	38.4676	38.7351	38.9991	39.2599	39.5175	39.772	40.0234	40.2719	40.5176
0.5	38.6305	38.9017	39.1692	39.4334	39.6942	39.9518	40.2064	40.4579	40.7065	40.9522

Valores de I_{ds} para valores de V_{gs} entre –2 y 0.5 y valores de V_{ds} entre 1.1 y 2

	2.1	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3**
-2	0	0	0	0	0	0	0	0	0	0
-1.9	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
-1.8	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
-1.7	0.0003	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0005	0.0005
-1.6	0.0008	0.0008	0.0009	0.0009	0.0009	0.001	0.001	0.001	0.001	0.0011
-1.5	0.0018	0.0019	0.002	0.002	0.0021	0.0022	0.0023	0.0023	0.0024	0.0025
-1.4	0.0042	0.0044	0.0045	0.0047	0.0049	0.005	0.0052	0.0053	0.0055	0.0057
-1.3	0.0096	0.01	0.0104	0.0108	0.0112	0.0115	0.0119	0.0123	0.0127	0.013
-1.2	0.022	0.0229	0.0238	0.0247	0.0256	0.0265	0.0274	0.0282	0.0291	0.0299
-1.1	0.0504	0.0525	0.0546	0.0566	0.0587	0.0606	0.0626	0.0646	0.0665	0.0684
-1	0.1147	0.1195	0.1242	0.1288	0.1334	0.138	0.1425	0.1469	0.1513	0.1557
-0.9	0.2574	0.268	0.2786	0.289	0.2993	0.3095	0.3196	0.3296	0.3394	0.3492
-0.8	0.5599	0.5831	0.606	0.6287	0.6511	0.6733	0.6952	0.7169	0.7384	0.7597
-0.7	1.1448	1.1922	1.2391	1.2854	1.3312	1.3766	1.4214	1.4658	1.5097	1.5531
-0.6	2.0993	2.1861	2.2719	2.3567	2.4406	2.5236	2.6057	2.687	2.7674	2.8469
-0.5	3.5674	3.7033	3.8376	3.9704	4.1017	4.2316	4.3602	4.4873	4.6131	4.7377
-0.4	6.6917	6.8717	7.0498	7.2258	7.3999	7.5721	7.7425	7.9111	8.0779	8.243
-0.3	11.133	11.3427	11.55	11.7551	11.9579	12.1586	12.357	12.5534	12.7477	12.94
-0.2	16.2503	16.4762	16.6996	16.9205	17.139	17.3551	17.5689	17.7804	17.9897	18.1968
-0.1	21.5685	21.8022	22.0334	22.2619	22.488	22.7116	22.9328	23.1516	23.3682	23.5825
0	26.6887	26.926	27.1607	27.3928	27.6223	27.8493	28.0739	28.2961	28.516	28.7336
0.1	31.2994	31.5383	31.7746	32.0082	32.2392	32.4678	32.6939	32.9176	33.1389	33.3579
0.2	35.218	35.4576	35.6945	35.9288	36.1605	36.3898	36.6165	36.8409	37.0628	37.2825
0.3	38.418	38.6579	38.8951	39.1297	39.3618	39.5913	39.8183	40.0429	40.2652	40.4852
0.4	40.7604	41.0004	41.2378	41.4725	41.7046	41.9343	42.1615	42.3862	42.6086	42.8287
0.5	41.195	41.4351	41.6726	41.9073	42.1395	42.3692	42.5965	42.8213	43.0437	43.2638

Valores de I_{ds} para valores de V_{gs} entre –2 y 0.5 y valores de V_{ds} entre 2.1 y 3

Si representamos estos datos gráficamente, tenemos la siguiente figura:

I_{ds} (mA)

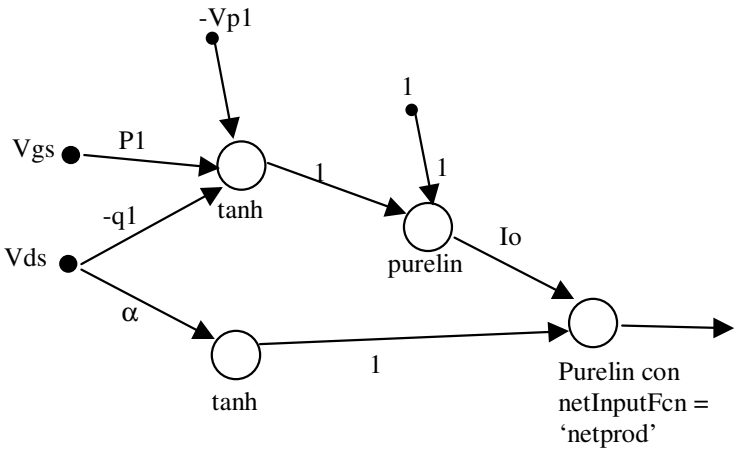
V_{ds} (V)

V_{gs} (V)

La expresión que tenemos que implementar mediante una red neuronal es la siguiente:

$$I_{ds} = I_0 \cdot \left[1 + tanh\left(p_1 \cdot V_{gs} - q_1 \cdot V_{ds} - V_{p1}\right)\right] \cdot \left[tanh\left(\alpha \cdot V_{ds}\right)\right]$$

Para implementar esa expresión utilizaremos la siguiente red neuronal:



Esta red neuronal presenta dos entradas y consta de cuatro neuronas, una de las cuales será la salida de la red. Dos de ellas se emplean en calcular las tangentes hiperbólicas, otra neurona sirve para sumar un uno a una de las tangentes y finalmente la última neurona servirá para realizar una multiplicación. En esta red neuronal fijaremos ciertos pesos a valores constantes para poder obtener la expresión deseada.

Tras entrenar la red, obtuvimos un error cuadrático medio de 0.3634, quedando los siguientes valores para los distintos parámetros de la red neuronal:

Io	p1	q1	Vp1	α
20.9457	2.7753	0.1474	0.0147	4.7323

Con lo que finalmente queda la siguiente expresión:

$$I_{ds} = 20.9457 \cdot \left[1 + tanh\left(2.7753 \cdot V_{gs} + 0.1474 \cdot V_{ds} - 0.0147\right)\right] \cdot \left[tanh\left(4.7323 \cdot V_{ds}\right)\right]$$

Finalmente, vamos a representar la salida de la red neuronal cuando le presentamos los mismos datos de entrada para ver gráficamente la bondad de la aproximación:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

Se puede ver como el resultado obtenido por la red neuronal es muy similar al deseado, por lo que el modelo refleja fielmente el comportamiento del transistor, aunque se observa que no se consigue la misma pendiente para V_{gs}= 0.5 V.

5.3.2 Transistor HEMT modelo foundry ED02AH con W=6x50

Los datos que emplearemos en el entrenamiento de la red neuronal se muestran en las siguientes tablas:

	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
-2	0	0	0	0	0	0	0	0	0	0.0001	0.0001
-1.9	0	0	0	0	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
-1.8	0	0	0.0001	0.0001	0.0001	0.0002	0.0002	0.0002	0.0002	0.0003	0.0003
-1.7	0	0.0001	0.0001	0.0002	0.0003	0.0004	0.0004	0.0005	0.0005	0.0006	0.0007

-1.6	0	0.0002	0.0003	0.0005	0.0006	0.0008	0.0009	0.0011	0.0012	0.0014	0.0015
-1.5	0	0.0004	0.0008	0.0011	0.0015	0.0018	0.0022	0.0025	0.0028	0.0032	0.0035
-1.4	0	0.0009	0.0017	0.0025	0.0034	0.0042	0.005	0.0057	0.0065	0.0073	0.008
-1.3	0	0.002	0.0039	0.0058	0.0077	0.0095	0.0114	0.0132	0.0149	0.0167	0.0184
-1.2	0	0.0045	0.009	0.0133	0.0176	0.0219	0.026	0.0302	0.0342	0.0382	0.0422
-1.1	0	0.0103	0.0205	0.0305	0.0403	0.05	0.0596	0.069	0.0783	0.0875	0.0966
-1	0	0.0235	0.0466	0.0693	0.0917	0.1138	0.1355	0.157	0.1782	0.1991	0.2197
-0.9	0	0.0526	0.1044	0.1554	0.2056	0.2552	0.304	0.3521	0.3996	0.4465	0.4928
-0.8	0	0.1145	0.2272	0.3381	0.4474	0.5551	0.6613	0.7661	0.8694	0.9714	1.072
-0.7	0	0.2344	0.465	0.6918	0.9152	1.1355	1.3526	1.5668	1.778	1.9865	2.1922
-0.6	0	0.4351	0.8609	1.2779	1.6876	2.091	2.4886	2.8808	3.2676	3.6493	4.0259
-0.5	0	1.1927	2.1763	2.9713	3.6672	4.3183	4.9476	5.5638	6.1701	6.7678	7.3574
-0.4	0	5.1477	8.6615	10.7074	12.0102	13.0102	13.8925	14.7261	15.5357	16.3301	17.1124
-0.3	0	12.0627	19.9146	23.9851	26.1497	27.5473	28.657	29.6566	30.6095	31.5383	32.4508
-0.2	0	20.7322	33.985	40.522	43.6789	45.4813	46.781	47.8942	48.9333	49.9382	50.9227
-0.1	0	30.0915	49.1578	58.3248	62.5125	64.7091	66.1705	67.3629	68.4523	69.4969	70.5173
0	0	39.2571	64.0091	75.7376	80.9178	83.4821	85.0833	86.3349	87.4552	88.5207	89.5585
0.1	0	47.5742	77.4826	91.5298	97.6036	100.4942	102.2148	103.5127	104.6537	105.7309	106.7772
0.2	0	54.6683	88.9736	104.9963	111.8296	114.9956	116.815	118.1494	119.3051	120.3895	121.4401
0.3	0	60.4719	98.3737	116.0117	123.4652	126.8552	128.7544	130.1174	131.284	132.3729	133.4261
0.4	0	64.7235	105.2601	124.0811	131.9885	135.5423	137.4994	138.8831	140.0571	141.1491	142.2037
0.5	0	65.5112	106.5359	125.5762	133.5679	137.1521	139.1201	140.5078	141.6834	142.7761	143.831

Valores de I_{ds} para valores de V_{gs} entre -2 y 0.5 y valores de V_{ds} entre 0 y 1

	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2**
-2	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
-1.9	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
-1.8	0.0003	0.0003	0.0004	0.0004	0.0004	0.0004	0.0005	0.0005	0.0005	0.0005
-1.7	0.0007	0.0008	0.0008	0.0009	0.001	0.001	0.0011	0.0011	0.0012	0.0012
-1.6	0.0017	0.0018	0.0019	0.0021	0.0022	0.0023	0.0025	0.0026	0.0027	0.0028
-1.5	0.0038	0.0041	0.0044	0.0048	0.0051	0.0054	0.0057	0.006	0.0063	0.0065
-1.4	0.0088	0.0095	0.0102	0.0109	0.0116	0.0123	0.013	0.0137	0.0144	0.015
-1.3	0.0201	0.0218	0.0234	0.0251	0.0267	0.0283	0.0299	0.0314	0.033	0.0345
-1.2	0.0461	0.05	0.0538	0.0575	0.0613	0.0649	0.0685	0.0721	0.0757	0.0792
-1.1	0.1055	0.1144	0.1231	0.1317	0.1402	0.1486	0.1569	0.1651	0.1732	0.1812
-1	0.2401	0.2602	0.28	0.2996	0.3189	0.3381	0.3569	0.3756	0.394	0.4123

-0.9	0.5384	0.5835	0.628	0.672	0.7154	0.7583	0.8006	0.8425	0.8839	0.9248
-0.8	1.1713	1.2694	1.3662	1.4618	1.5563	1.6496	1.7417	1.8328	1.9228	2.0117
-0.7	2.3953	2.5957	2.7936	2.9891	3.1821	3.3728	3.5612	3.7474	3.9314	4.1132
-0.6	4.3977	4.7647	5.1271	5.485	5.8384	6.1876	6.5325	6.8734	7.2103	7.5432
-0.5	7.9394	8.5139	9.0811	9.6412	10.1945	10.741	11.2809	11.8144	12.3417	12.8628
-0.4	17.8841	18.6458	19.3977	20.1403	20.8737	21.5981	22.3139	23.0211	23.7201	24.4109
-0.3	33.3501	34.2375	35.1134	35.9784	36.8327	37.6766	38.5104	39.3342	40.1484	40.9531
-0.2	51.8919	52.8479	53.7916	54.7233	55.6435	56.5526	57.4507	58.3381	59.2151	60.0819
-0.1	71.5207	72.5101	73.4866	74.4507	75.4029	76.3434	77.2727	78.1909	79.0983	79.9952
0	90.5779	91.5826	92.5741	93.553	94.5198	95.4748	96.4183	97.3505	98.2719	99.1825
0.1	107.804	108.8157	109.8139	110.7993	111.7726	112.734	113.6838	114.6223	115.5498	116.4665
0.2	122.4705	123.4852	124.4864	125.4748	126.4509	127.4151	128.3676	129.3089	130.2391	131.1585
0.3	134.4581	135.4743	136.4768	137.4664	138.4438	139.4092	140.363	141.3055	142.2368	143.1574
0.4	143.2365	144.2534	145.2565	146.2467	147.2246	148.1905	149.1448	150.0878	151.0197	151.9408
0.5	144.8642	145.8813	146.8846	147.8751	148.8532	149.8194	150.774	151.7172	152.6493	153.5706

Valores de I_{ds} para valores de V_{gs} entre –2 y 0.5 y valores de V_{ds} entre 1.1 y 2.1

	2.1	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3**
-2	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
-1.9	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
-1.8	0.0006	0.0006	0.0006	0.0006	0.0007	0.0007	0.0007	0.0007	0.0007	0.0008
-1.7	0.0013	0.0014	0.0014	0.0015	0.0015	0.0016	0.0016	0.0017	0.0017	0.0018
-1.6	0.003	0.0031	0.0032	0.0033	0.0035	0.0036	0.0037	0.0038	0.0039	0.004
-1.5	0.0068	0.0071	0.0074	0.0077	0.0079	0.0082	0.0085	0.0087	0.009	0.0093
-1.4	0.0157	0.0163	0.017	0.0176	0.0182	0.0188	0.0195	0.0201	0.0207	0.0213
-1.3	0.036	0.0375	0.039	0.0404	0.0419	0.0433	0.0447	0.0461	0.0475	0.0488
-1.2	0.0826	0.086	0.0894	0.0928	0.0961	0.0994	0.1026	0.1058	0.109	0.1121
-1.1	0.1891	0.197	0.2047	0.2124	0.2199	0.2274	0.2348	0.2422	0.2494	0.2566
-1	0.4303	0.4481	0.4657	0.4831	0.5004	0.5174	0.5343	0.551	0.5675	0.5838
-0.9	0.9652	1.0052	1.0447	1.0837	1.1224	1.1606	1.1984	1.2359	1.2729	1.3095
-0.8	2.0997	2.1866	2.2726	2.3576	2.4417	2.5248	2.6071	2.6885	2.769	2.8487
-0.7	4.293	4.4707	4.6464	4.8202	4.9921	5.1621	5.3303	5.4967	5.6613	5.8243
-0.6	7.8723	8.1977	8.5194	8.8376	9.1523	9.4636	9.7715	10.0762	10.3776	10.6759
-0.5	13.3779	13.8872	14.3908	14.8889	15.3814	15.8686	16.3506	16.8274	17.2993	17.7662
-0.4	25.0938	25.769	26.4366	27.0968	27.7497	28.3956	29.0345	29.6666	30.2921	30.9111
-0.3	41.7486	42.535	43.3127	44.0817	44.8423	45.5946	46.3389	47.0752	47.8038	48.5249
-0.2	60.9388	61.7859	62.6236	63.4519	64.2712	65.0816	65.8833	66.6765	67.4613	68.238

-0.1	80.8819	81.7584	82.6251	83.4822	84.33	85.1685	85.998	86.8187	87.6308	88.4344
0	100.0827	100.9727	101.8527	102.7229	103.5836	104.4349	105.2771	106.1104	106.9349	107.7509
0.1	117.3728	118.2687	119.1546	120.0307	120.8971	121.7542	122.602	123.4409	124.2709	125.0923
0.2	132.0674	132.9659	133.8544	134.7331	135.602	136.4616	137.3119	138.1532	138.9857	139.8095
0.3	144.0674	144.9671	145.8567	146.7365	147.6066	148.4672	149.3186	150.161	150.9945	151.8194
0.4	152.8513	153.7515	154.6416	155.5219	156.3924	157.2536	158.1054	158.9483	159.7823	160.6076
0.5	154.4813	155.3818	156.2721	157.1525	158.0233	158.8846	159.7367	160.5797	161.4139	162.2394

Valores de I_{ds} para valores de V_{gs} entre –2 y 0.5 y valores de V_{ds} entre 2.1 y 3.1

Si representamos estos datos gráficamente, se puede observar mejor el comportamiento del transistor a modelar:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

Tras proceder de la misma manera que en el apartado anterior, obtenemos un error cuadrático medio de 5.1102 y los siguientes valores para los parámetros del modelo en estudio:

Io	**p1**	**q1**	**Vp1**	**α**
78.5463	2.7753	0.1474	0.0147	4.7323

Con lo que finalmente queda la siguiente expresión:

$$I_{ds} = 78.5463 \cdot \left[1 + tanh\left(2.7753 \cdot V_{gs} + 0.1474 \cdot V_{ds} - 0.0147\right)\right] \cdot \left[tanh\left(4.7323 \cdot V_{ds}\right)\right]$$

A continuación se muestra la representación gráfica de los resultados obtenidos:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

De nuevo se comprueba que el modelo refleja fielmente el comportamiento del transistor en estudio, con el mismo problema a la hora de conseguir la misma pendiente de la I_{ds} respecto a la V_{ds} para V_{gs} = 0.5 V.

5.4 Aproximación de la curva I_{ds} frente a V_{gs}, V_{ds} (Modelo de Angelov)

En este apartado vamos a estudiar otro modelo matemático, el modelo de Angelov, mediante el cual trataremos de nuevo obtener una expresión para la intensidad de drenador en función de la tensión de puerta y de la tensión de drenador. Vamos a utilizar de nuevo los datos correspondientes a los transistores HEMT modelo foundry ED02AH con W = 2x40 y con W = 6x50.

El modelo de Angelov, cuyo objetivo es modelar dispositivos MESFET y HEMT, permite la extracción de los parámetros de éste por simple inspección de las características de DC obtenidas experimentalmente , véase I_{ds} (V_{gs},V_{ds}) y gm (V_{gs}), con lo cual se modelará la I_{ds} y sus derivadas con buena precisión.

En este modelo, la función de la corriente de drenador se describe mediante la siguiente expresión:

$$I_{ds} = I_{pk}\left[1 + tanh(\varphi)\right] \cdot \left(1 + \lambda \cdot V_{ds}\right) \cdot tanh(\alpha \cdot V_{ds})$$

donde I_{ds} es la corriente de drenador a la que se tiene la mayor transconductancia, λ es el parámetro de modulación de la longitud del canal y α es el parámetro de tensión de saturación. φ es en general una función de series de potencia centrada en V_{pk} con V_{gs} como variable:

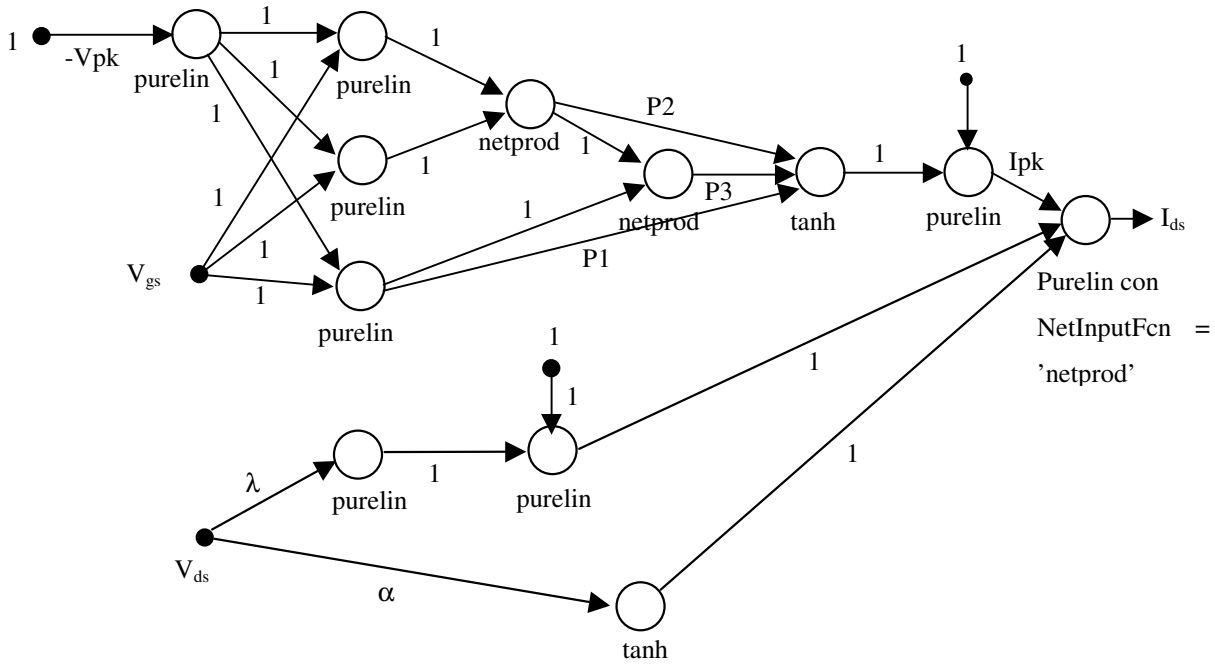
$$\varphi = P_1 \cdot \left(V_{gs} - V_{pk}\right) + P_2 \cdot \left(V_{gs} - V_{pk}\right)^2 + P_3 \cdot \left(V_{gs} - V_{pk}\right)^3 + \ldots$$

donde V_{pk} es la tensión de puerta para la transconductancia máxima. Esta función I_{ds} seleccionada tiene derivadas bien definidas y una de las ventajas de este modelo es su simplicidad.

Ya tenemos bien definido el modelo. Ahora se trata de crear una red neuronal representación del modelo y entrenarla con los datos experimentales de forma que obtengamos los parámetros del modelo para el transistor en cuestión. La expresión que debe realizar la red neuronal es la siguiente:

$$I_{ds} = I_{pk}\left[1 + tanh\left(P_1 \cdot \left(V_{gs} - V_{pk}\right) + P_2 \cdot \left(V_{gs} - V_{pk}\right)^2 + P_3 \cdot \left(V_{gs} - V_{pk}\right)^3 + ...\right)\right] \cdot \left(1 + \lambda \cdot V_{ds}\right) \cdot tanh\left(\alpha \cdot V_{ds}\right)$$

A continuación se muestra la red neuronal creada para estudiar este modelo:



La neurona a la salida de esta red realiza un producto de los tres términos de los que se compone la expresión del modelo, productos calculados por las neuronas que la preceden.

Vamos a proceder a hacer el análisis de nuevo con los dos transistores.

5.4.1 Transistor HEMT modelo foundry ED02AH con W=2x40

Tras entrenar la red neuronal, obtuvimos los siguientes valores para los distintos parámetros del modelo:

I_{pk}	V_{pk}	λ	α	P1	P2	P3
17.9431	-0.09322	0.0923	5.2322	2.5839	-0.3381	2.0071

Con estos parámetros se obtiene una error cuadrático medio de 0.271, quedando la expresión del modelo de la siguiente manera:

$$I_{ds} = 17.9431 \cdot \left[1 + tanh\left(2.5839 \cdot \left(V_{gs} + 0.09322\right) - 0.3381 \cdot \left(V_{gs} + 0.09322\right)^2 + 2.0071 \cdot \left(V_{gs} + 0.09322\right)^3\right)\right] \cdot \left(1 + 0.0923 \cdot V_{ds}\right) \cdot tanh\left(5.2322 \cdot V_{ds}\right)$$

A continuación se muestran los resultados obtenidos:

I_{ds} (mA)

V_{ds} (V)

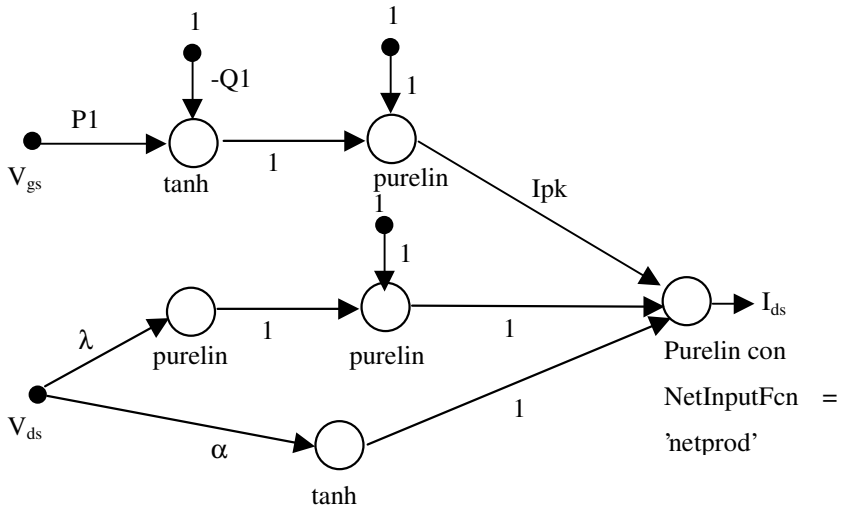
V_{gs} (V)

Con este modelo se mejora el resultado obtenido con el modelo del apartado anterior, pero este modelo parte con ventaja, ya que dispone de siete parámetros ajustables en lugar de cinco. Por tanto, vamos a nivelar el número de parámetros ajustables en ambos modelos cogiendo de la serie de potencias únicamente el término de primero orden, es decir, vamos a simplificar el modelo de la siguiente manera:

$$I_{ds} = I_{pk}\left[1 + tanh\left(P_1 \cdot \left(V_{gs} - V_{pk}\right)\right)\right] \cdot \left(1 + \lambda \cdot V_{ds}\right) \cdot tanh\left(\alpha \cdot V_{ds}\right) =$$
$$= I_{pk}\left[1 + tanh\left(P_1 \cdot V_{gs} - Q_1\right)\right] \cdot \left(1 + \lambda \cdot V_{ds}\right) \cdot tanh\left(\alpha \cdot V_{ds}\right)$$

donde hemos llamado Q_1 a $P_1 \cdot V_{pk}$.

Para simular este modelo mediante una red neuronal, se tiene un esquema más sencillo que en el caso anterior:



Tras entrenar esta red neuronal, los valores que se obtuvieron para los distintos parámetros del modelo fueron los siguientes:

I_{pk}	V_{pk}	λ	α	P1
17.9587	-0.08780	0.09284	5.2398	2.8073

Quedando la siguiente expresión:

$$I_{ds} = 17.9587 \cdot \left[1 + tanh\left(2.8073 \cdot \left(V_{gs} + 0.0878\right)\right)\right] \cdot \left(1 + 0.09284 \cdot V_{ds}\right) \cdot tanh\left(5.2398 \cdot V_{ds}\right)$$

Para estos valores se obtiene un error cuadrático medio de 0.3642, valor lógicamente peor que para el modelo con la serie de potencias hasta orden tres, ya que en este caso se tienen dos parámetros ajustables menos. Si comparamos el valor conseguido con el modelo del apartado anterior que también tenía cinco parámetros ajustables, vemos que se tienen valores muy parecidos. A continuación se muestra una gráfica con la salida de la red neuronal para distintos valores de V_{gs} y V_{ds}:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

5.4.2 Transistor HEMT modelo foundry ED02AH con W=6x50

Si procedemos de la misma manera con este transistor, obtenemos los siguientes valores para los distintos parámetros para el caso del modelo de orden tres (siete parámetros):

$\mathbf{I_{pk}}$	$\mathbf{V_{pk}}$	$\boldsymbol{\lambda}$	$\boldsymbol{\alpha}$	**P1**	**P2**	**P3**
67.3519	-0.0929	0.09219	5.2301	2.5835	-0.3475	1.9741

Obteniéndose un error cuadrático medio de 3.81168. La expresión que queda para este modelo es la siguiente:

$$I_{ds} = 67.3519 \cdot \left[1 + tanh\left(2.5835 \cdot \left(V_{gs} + 0.0929\right) - 0.3475 \cdot \left(V_{gs} + 0.0929\right)^2 + 1.9741 \cdot \left(V_{gs} + 0.0929\right)^3\right)\right] \cdot \left(1 + 0.09219 \cdot V_{ds}\right) \cdot tanh\left(5.2301 \cdot V_{ds}\right)$$

A continuación se muestran los valores obtenidos por la red neuronal:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

Si, al igual que hicimos con el otro transistor, simplificamos el modelo de siete a cinco parámetros, obtenemos los siguientes resultados:

$\mathbf{I_{pk}}$	$\mathbf{V_{pk}}$	$\boldsymbol{\lambda}$	$\boldsymbol{\alpha}$	**P1**
67.345	-0.0878	0.09284	5.2398	2.8073

Quedando la siguiente expresión:

$$I_{ds} = 67.345 \cdot \left[1 + tanh\left(2.8073 \cdot \left(V_{gs} + 0.0878\right)\right)\right] \cdot \left(1 + 0.09284 \cdot V_{ds}\right) \cdot tanh\left(5.2398 \cdot V_{ds}\right)$$

Con estos valores se obtuvo un error cuadrático medio de 5.12203. A continuación se muestra gráficamente el comportamiento del modelo obtenido para este caso:

I_{ds} (mA)

V_{ds} (V)

V_{gs} (V)

5.5 Aproximación de la curva G_{ds} frente a V_{gs}, V_{ds} (Modelo de Fujii)

En este apartado vamos a comprobar el funcionamiento de un modelo para la caracterización de la G_{ds} de un transistor HEMT, como es el modelo de Fujii, aunque este modelo está originariamente pensado para transistores MESFET.

La función de modelado propuesta por Fujii es la siguiente:

$$G_{ds}\left(V_{gs}, V_{ds}\right) = F_1\left(V_{gs}, 0\right) \cdot F_2\left(V_{gs}, V_{ds}\right)$$

Para F_1, la función tangente hiperbólica representa bien la dependencia de G_{ds} respecto a V_{gs} y sus derivadas en $V_{ds} = 0$. F_1 viene dada por:

$$F_1\left(V_{gs}\right) \approx F_{cnt} \cdot \left[1 + tanh(\varphi)\right]$$

donde F_{cnt} es un valor del modelo en el centro de la región lineal, y el valor de V_{gs} en F_{cnt} es V_{cnt}. φ es una función de series de potencia centrada en F_{cnt} con V_{gs} como variable, y viene dada por:

$$\varphi = P_1 \cdot \left(V_{gs} - V_{pk}\right) + P_2 \cdot \left(V_{gs} - V_{pk}\right)^2 + P_3 \cdot \left(V_{gs} - V_{pk}\right)^3 + \ldots$$

donde P_n, n = 1,2,3,... son los parámetros ajustables.

La función F_2 representa la dependencia de G_{ds} respecto a V_{ds} y sus derivadas en cada valor de la tensión de puerta , y viene representada por:

$$F_2(V_{gs}, V_{ds}) \approx 1 - \left[A_1(V_{gs}) \cdot |V_{ds}| + A_2(V_{gs}) \cdot |V_{ds}|^2 + A_3(V_{gs}) \cdot |V_{ds}|^3 + ...\right]$$

donde $A_n(V_{gs})$, n = 1,2,3,... son los coeficientes de ajuste dependientes de V_{gs}, que tienen la siguiente expresión:

$$A_n(V_{gs}) \approx F_{cnt} \cdot \left[(1 + \lambda \cdot V_{gs}) + tanh(\varphi)\right]$$

Una vez visto el modelo, vamos a intentar hallar los distintos parámetros de éste para ajustarse a los valores de nuestros transistores en estudio. Debido a la complejidad del modelo a la hora de implementarlo mediante una red neuronal, vamos a proceder al ajuste de los parámetros ayudándonos de una función implementada en Matlab, denominada 'fmin', orientada a la minimización de funciones.

5.5.1 Transistor HEMT modelo foundry ED02AH con W=2x40

A continuación se muestran los valores de G_{ds} en función de V_{gs} y V_{ds} para este transistor, valores que utilizaremos para intentar hallar los valores de los parámetros de este modelo:

G_{ds}

V_{ds} (V)

V_{gs} (V)

Tras proceder a ajustar los parámetros, se obtuvieron los siguientes resultados:

Parámetros	Parámetros de F_1	Parámetros de A_1	Parámetros de A_2	Parámetros de A_3
$\mathbf{F_{cnt}}$	72.3347	1.8228	1.0727	0.1866
$\mathbf{P_1}$	2.5722	0.0005	0.2730	-0.0935
$\mathbf{P_2}$	-0.3162	0.1463	0.0760	-0.0231
$\mathbf{P_3}$	10.0462	0.0108	1.4973	3.3724
$\mathbf{V_{cnt}}$	-0.0575	0.0293	0.0759	0.1230
$\boldsymbol{\lambda}$	-	0.2483	0.0079	0.2893

A continuación se muestran los resultados obtenidos gráficamente:

G_{ds}

V_{ds} (V)

V_{gs} (V)

Si comparamos los resultados obtenidos con los que se deberían obtener, se puede ver que este modelo no es capaz de aproximar correctamente la función G_{ds} de este transistor.

5.5.2 Transistor HEMT modelo foundry ED02AH con W=6x50

A continuación se muestran los valores de G_{ds} en función de V_{gs} y V_{ds} para este transistor:

G_{ds}

V_{ds} (V)

V_{gs} (V)

Procediendo de la misma manera con el ajuste los parámetros, se obtuvieron los siguientes resultados:

Parámetros	Parámetros de F_1	Parámetros de A_1	Parámetros de A_2	Parámetros de A_3
$\mathbf{F_{cnt}}$	281.2621	32.0916	-1.2931	0.2240
$\mathbf{P_1}$	2.8650	0.005392	0.1049	0.2424
$\mathbf{P_2}$	-0.2141	-0.0881	-0.0218	-0.0619
$\mathbf{P_3}$	3.8656	-0.0135	-0.0196	0.1001
$\mathbf{V_{cnt}}$	-0.06129	-0.1055	0.0816	-0.2296
$\boldsymbol{\lambda}$	-	0.1448	0.0245	-0.1013

A continuación se muestran los resultados obtenidos gráficamente:

G_{ds}

V_{ds} (V)

V_{gs} (V)

Si comparamos los resultados obtenidos con los que se deberían obtener, de nuevo se vuelve a comprobar que este modelo no es capaz de aproximar correctamente la función G_{ds} de este transistor.

5.6 Evaluación de los resultados obtenidos

5.6.1 Aproximaciones de I_{ds} y g_m. Derivación e integración

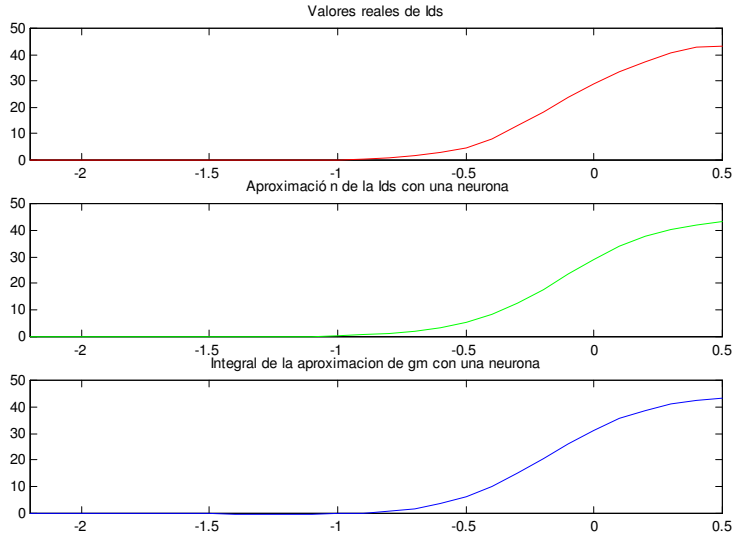
En los dos primeros apartados de este capítulo hemos utilizado redes neuronales para obtener una aproximación de las funciones de I_{ds} y g_m en función de la tensión V_{gs}, aproximación que se traduce en una expresión matemática implementada por la red neuronal. Como ya sabemos, la transconductancia g_m no es más que la derivada de la intensidad de drenador respecto a la tensión de puerta V_{gs}. Por tanto, si derivamos el resultado obtenido para la intensidad, deberíamos obtener una aproximación de la transconductancia. Así mismo, a partir de la aproximación obtenida para la transconductancia podríamos obtener una aproximación de la intensidad de drenador simplemente integrando respecto a la tensión de puerta. En este apartado vamos a comprobar qué resultados se obtienen si derivamos la aproximación de la intensidad y si integramos la aproximación de la transconductancia.

5.6.1.1 Integración de la transconductancia

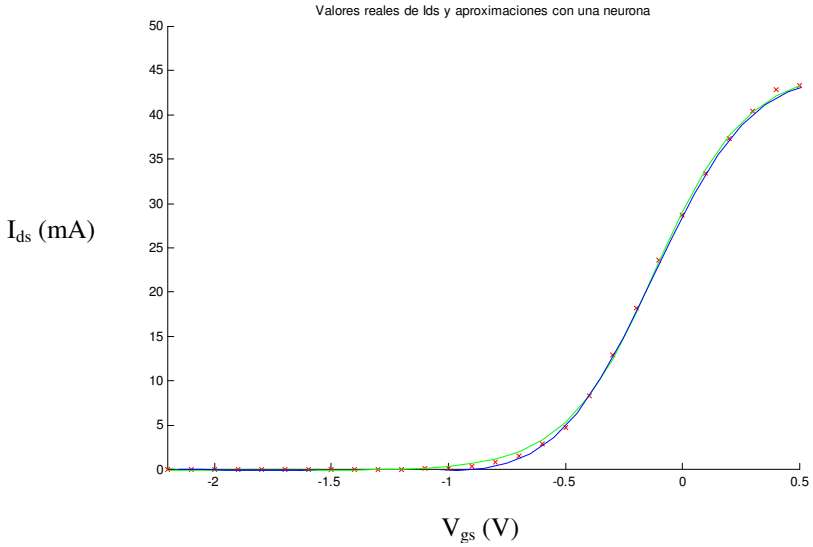
Vamos a ver cómo de buena es la aproximación que se obtiene de la intensidad de drenador obteniéndola como integral de la aproximación de la transconductancia obtenida mediante una red neuronal. Para ello nos centraremos únicamente en el caso del modelo foundry ED02AH con W=2x40 y Vds = 3V.

5.6.1.1.1 Empleando una neurona

En la siguiente gráfica se muestra la función de la intensidad de drenador a aproximar, la aproximación de dicha función mediante una red neuronal de una neurona y la integral de la aproximación de la transconductancia obtenida también mediante una red neuronal:



En la siguiente gráfica están representadas las tres curvas superpuestas, para poder apreciar la exactitud de las aproximaciones:

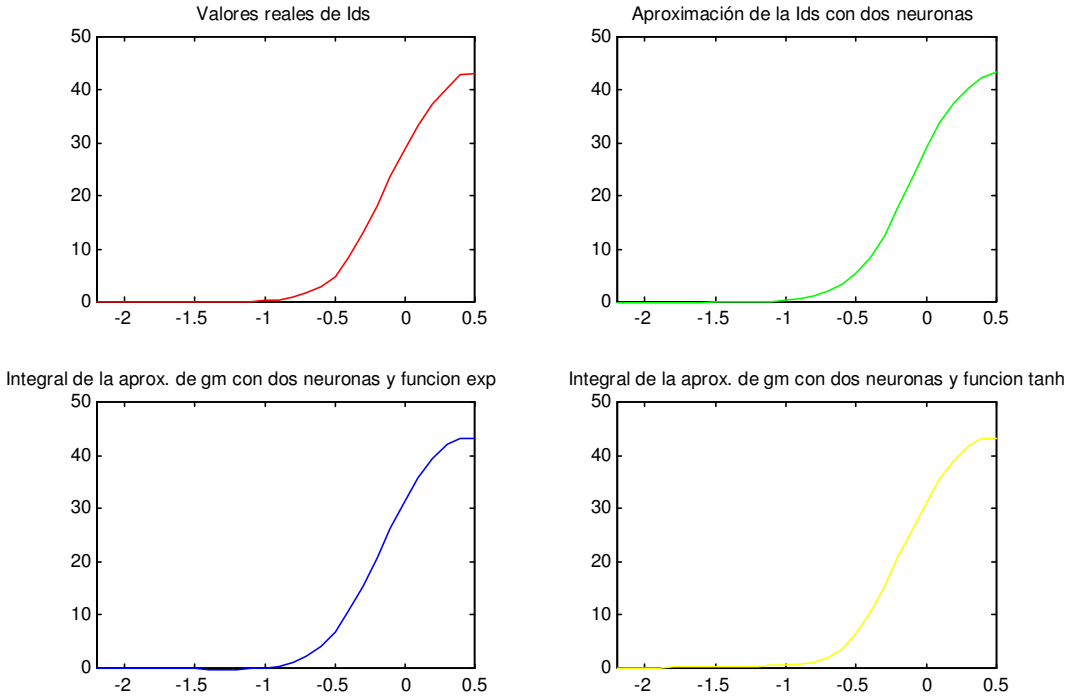


Como se puede ver, se consigue tan buena aproximación de la intensidad de drenador al integrar la aproximación de la transconductancia que aproximando directamente la intensidad de drenador.

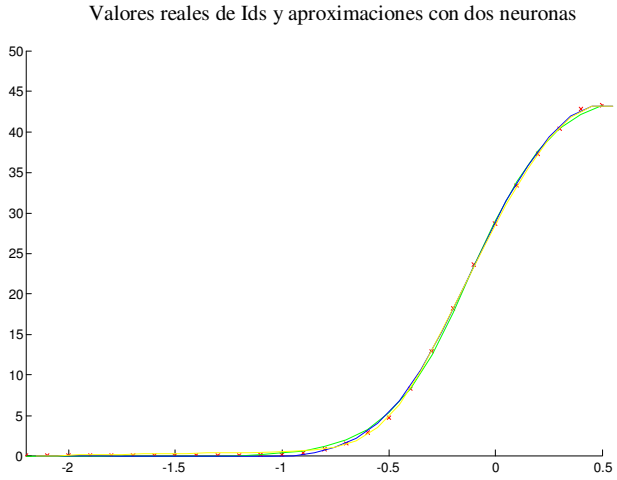
5.6.1.1.2 Empleando dos neuronas

En este caso en que se emplean dos neuronas, así como en el caso en que se emplean tres neuronas, que veremos más adelante, vamos a evaluar la integral de las dos aproximaciones de la transconductancia que se hallaron mediante del empleo de dos funciones de transferencia distintas para las neuronas situadas en la capa oculta de la red neuronal: la función exponencial cuadrática y la función tangente hiperbólica.

En la siguiente gráfica se pueden observar los resultados obtenidos tras aproximar directamente la intensidad de drenador con dos neuronas así como los obtenidos al integrar las dos aproximaciones de la transconductancia.



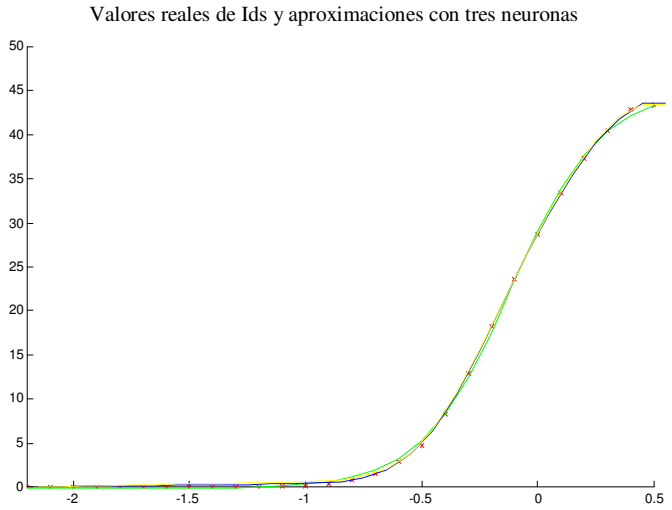
En la siguiente gráfica se encuentran representadas las cuatro curvas juntas para poder compararlas mejor:



De nuevo aquí se comprueba que se obtiene prácticamente el mismo resultado aproximando directamente la intensidad que integrando cualquiera de las aproximaciones de la transconductancia.

5.6.1.1.3 Empleando tres neuronas

En este caso se vuelve a repetir lo expuesto en los dos apartados anteriores. En la siguiente gráfica se vuelve a comprobar que se obtiene muy buenos resultados al integrar las aproximaciones de la transconductancia:



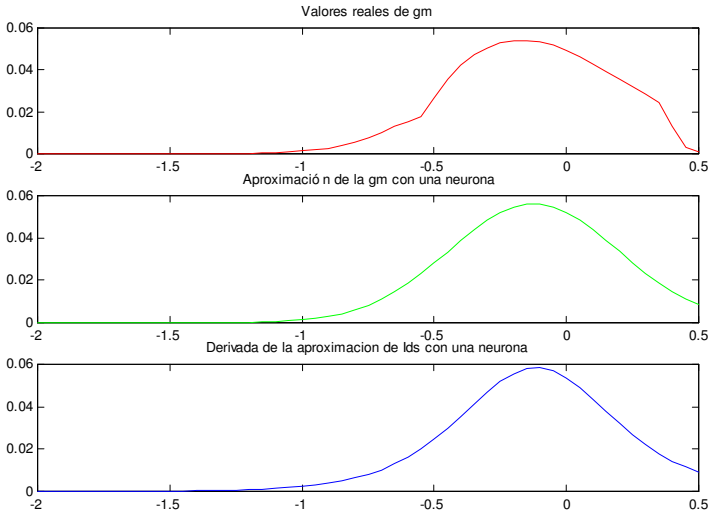
Decir que para el otro modelo de transistor estudiado se comprobó el mismo comportamiento.

5.6.1.2 Derivada de la intensidad de drenador

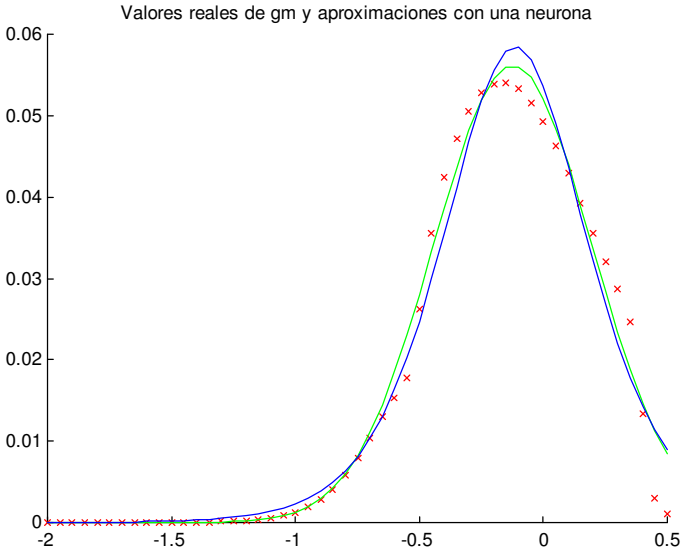
En este caso vamos a ver cómo de buena es la aproximación que se obtiene de la transconductancia obteniéndola como la derivada de la aproximación de la intensidad de drenador obtenida mediante una red neuronal. Para ello nos centraremos de nuevo únicamente en el caso del modelo foundry ED02AH con W=2x40 y V_{ds} = 3V.

5.6.1.2.1 Empleando una neurona

En la siguiente gráfica se muestra la función de la transconductancia a aproximar, la aproximación de dicha función mediante una red neuronal de una neurona y la derivada de la aproximación de la intensidad de drenador obtenida también mediante una red neuronal:



Se puede ver cómo no se obtienen resultados excesivamente buenos de ninguna de las dos maneras. En la siguiente gráfica, en la que representamos juntas las dos curvas y los puntos de los datos experimentales se puede apreciar mejor el comportamiento de ambos métodos:



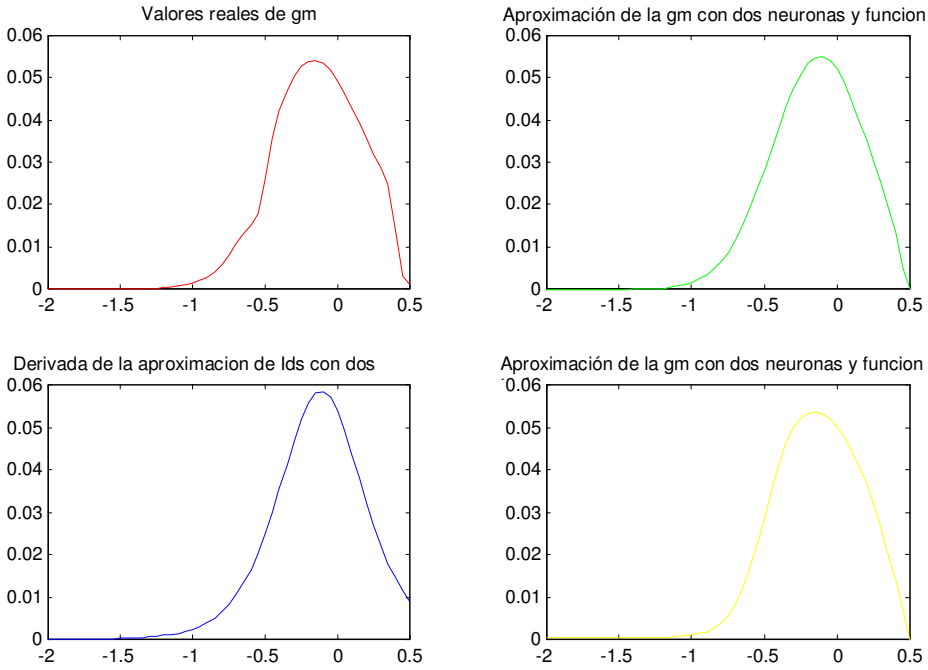
En dicha gráfica se aprecia cómo se obtiene un mejor resultado aproximando la transconductancia directamente que derivando la aproximación de la intensidad de drenador.

5.6.1.2.2 Empleando dos neuronas

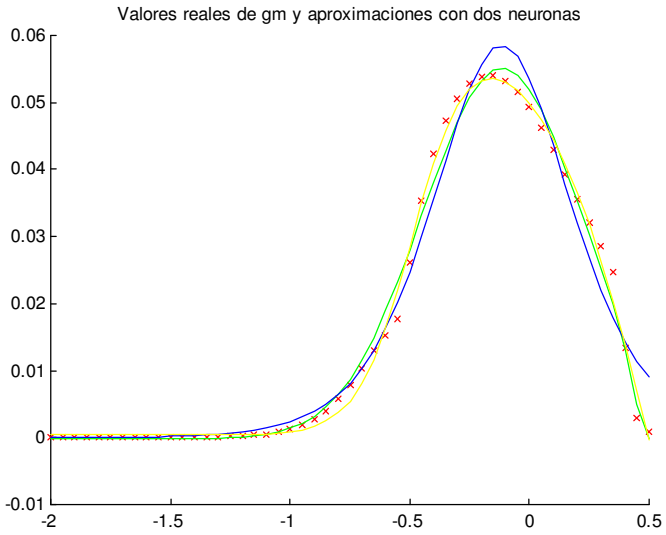
En este caso en que se emplean dos neuronas, así como en el caso en que se emplean tres neuronas, que veremos más adelante, vamos a evaluar la derivada de las aproximaciones de la intensidad de drenador

que se hallaron mediante del empleo de la función de transferencia tangente hiperbólica, así como también para el caso en el que se emplee la función de transferencia exponencial cuadrática.

En la siguiente gráfica se pueden observar los resultados obtenidos tras aproximar directamente la transconductancia con dos neuronas así como los obtenidos al integrar las dos aproximaciones de la transconductancia:



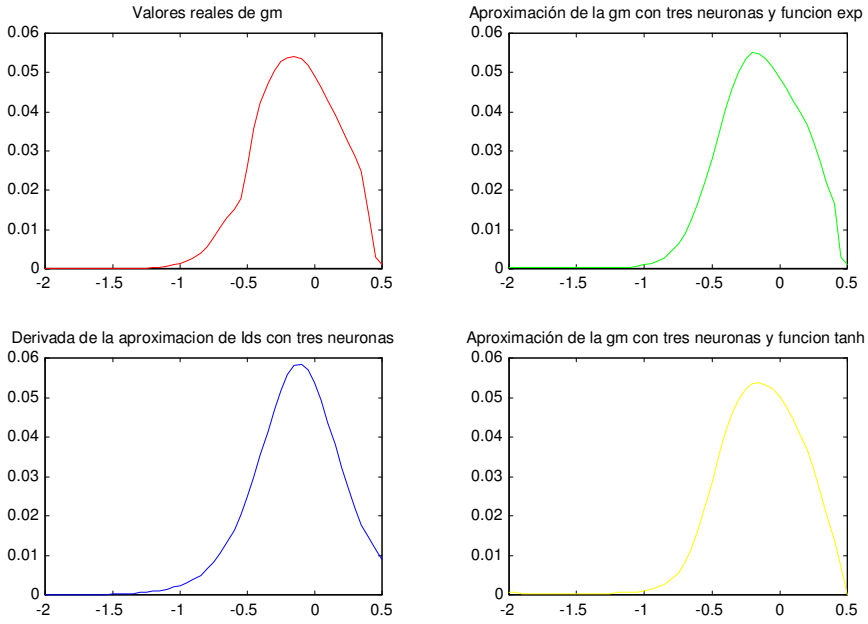
Si representamos los datos experimentales empleados para el entrenamiento junto con las tres curvas halladas:



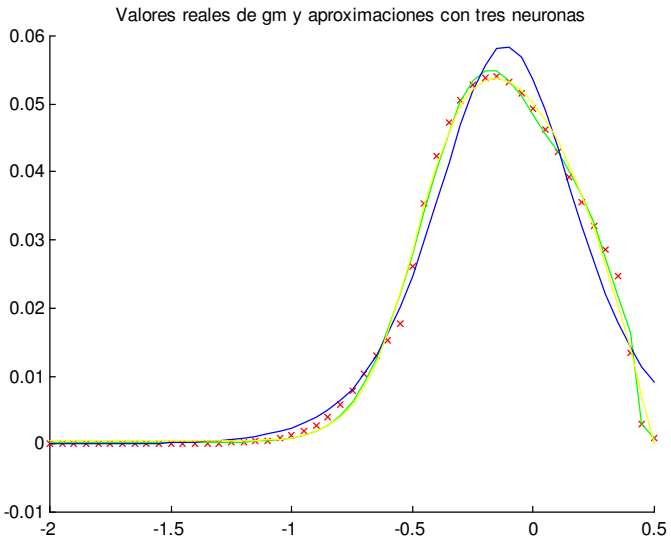
Como se puede ver, la curva de la derivada de la aproximación de I_{ds} con dos neuronas, que es la curva azul, es la que peor comportamiento presenta, mientras que las curvas de las aproximaciones de la g_m tanto con función tangente hiperbólica como con exponenciales cuadráticas siguen con más precisión los valores experimentales.

5.6.1.2.3 Empleando tres neuronas

En este caso se vuelve a repetir lo expuesto en los dos apartados anteriores. En la siguiente gráfica se vuelve a comprobar que se obtienen resultados sensiblemente peores al derivar las aproximaciones de la intensidad de drenador.



Si representamos las curvas juntas se puede apreciar mejor lo dicho anteriormente:



Para el otro modelo de transistor se repite exactamente el mismo comportamiento, de ahí que no presentemos los resultados obtenidos.

De todo lo expuesto, se puede concluir que, si queremos obtener una expresión matemática que caracterice correctamente las curvas de la intensidad de drenador y de la transconductancia del transistor, debemos aproximar la curva de la transconductancia y no la de la intensidad de drenador, ya que al integrar los resultados de la transconductancia se obtienen unos valores muy buenos para la intensidad de drenador, no sucediendo lo mismo en el sentido contrario, es decir, aproximando la intensidad de drenador y derivando ésta para obtener la transconductancia.

5.6.2 Aproximación de I_{ds} frente a V_{gs} y V_{ds}

En el primer apartado de este capítulo hemos visto unos modelos de redes neuronales para aproximar la función de la intensidad de drenador. Estos modelos no tratan de ser una representación del comportamiento de los transistores en estudio, es decir, no se trata de obtener un modelo en sí de estos transistores, sino lo que se trata es de aproximar una función sin tener en cuenta las características y fundamentos del comportamiento de los transistores; es por ello que estos estudios únicamente sirven para obtener una expresión matemática sencilla que permita obtener fácilmente los valores de la función aproximada, o lo que es lo mismo, interpolar a partir de resultados experimentales. En cambio, en los apartados 3 y 4 de este capítulo, hemos visto dos modelos que sí tratan de reflejar las características y el comportamiento de los transistores, y lo que hemos tratado es de averiguar los parámetros de los que se componen estos modelos para los transistores en estudio. Hemos visto que el modelo de Angelov se comporta de mejor manera que el modelo visto en el apartado 3, que hemos llamado Modelo 1, pero hay que decir que el modelo de Angelov dispone de 7 parámetros ajustables en lugar de 5. Si en el modelo de Angelov se reducen los parámetros libres de 7 a 5 limitando el orden de la serie de potencias vimos cómo

el comportamiento de ambos modelos se igualaban. Decir que el resultado de ambos modelos es bastante bueno, como se pudo comprobar en los apartados anteriores.

5.6.3 Aproximación de G_{ds} frente a V_{gs} y V_{ds}

En este caso hemos tratado de utilizar el modelo de Fujii para modelar el comportamiento de la G_{ds} de los transistores estudiados. Este modelo ha sido elaborado para transistores MESFET, y a la hora de emplearlos para modelar nuestros dos transistores HEMT no se han comportado correctamente, de ahí que desechemos su utilización.

5.6.4 Conclusiones

Por tanto, para concluir, si queremos obtener una expresión matemática sencilla para obtener los valores de la intensidad de drenador o de la transconductancia en función de la tensión de puerta, deberemos emplear uno de los dos modelos seleccionados en los dos primeros apartados. A continuación vamos a mostrar los valores obtenidos del error cuadrático medio para cada una de las aproximaciones:

I_{ds}	**W = 2x40, V_{ds} = 3 V**	**W = 6x50, V_{ds} = 1.4 V**
1 neurona	0.0977493	1.69802
2 neuronas	0.0977493	1.69802
3 neuronas	0.0977493	1.69802

G_m	**W = 2x40, V_{ds} = 3 V**	**W = 6x50, V_{ds} = 1.4 V**
1 neurona (tanh)	$6.20138 \cdot 10^{-6}$	$1.07539 \cdot 10^{-4}$
2 neuronas (exponencial cuadrática)	$3.64444 \cdot 10^{-6}$	$3.725 \cdot 10^{-5}$
2 neuronas (tanh)	$2.00506 \cdot 10^{-6}$	$2.24555 \cdot 10^{-5}$
3 neuronas (exponencial cuadrática)	$1.34746 \cdot 10^{-6}$	$1.99862 \cdot 10^{-5}$
3 neuronas (tanh)	$2.00506 \cdot 10^{-6}$	$2.24523 \cdot 10^{-5}$

Como se puede ver, cuando se trata de ajustar o encontrar una expresión matemática sencilla para I_{ds} frente a V_{gs}, lo mejor es utilizar el modelo de una neurona con función de activación tangente hiperbólica, ya que la inclusión de más neuronas no contribuye significativamente a una mejora en la aproximación, ya que la propia forma de la función tangente hiperbólica se ajusta perfectamente a los valores que se

tienen para estos transistores. Entre ambos transistores no se puede realizar una comparación ya que presentan distintos rangos de valores para la intensidad de drenador.

Si se quiere obtener una buena aproximación de ambos, se deberá aproximar la transconductancia, ya que su derivada sirve de buena aproximación de la intensidad de drenador, mientras que la derivada de la intensidad de drenador no es tan exacta como aproximación de la transconductancia.

Si lo que queremos es obtener un modelo que refleje bien el comportamiento y funcionamiento de los transistores, deberemos usar uno de los dos modelos vistos en los apartados 3 y 4. Con estos modelos se obtienen expresiones matemáticas derivables de las que se pueden obtener otros parámetros y figuras características de este tipo de transistores, así como simular el comportamiento de diferentes circuitos no lineales como mezcladores con un alto grado de exactitud. A continuación se muestran los valores obtenidos en términos del error cuadrático medio:

I_{ds}	W = 2x40	W = 6x50
Modelo GSR1	0.3634	5.1102
Modelo de Angelov (7 parámetros)	0.2710	3.8117
Modelo de Angelov (5 parámetros)	0.3642	5.1220

Como se puede ver en la tabla, con el modelo de Angelov se puede conseguir mejores resultados aumentando el orden de la serie de potencias (y por tanto aumentando el número de parámetros ajustables).

6 APLICACIÓN DE LAS REDES NEURONALES PARA LA PREDICCIÓN DE LAS PÉRDIDAS BÁSICAS DE POPAGACIÓN

6.1 Fundamentos

Existen multitud de métodos para la predicción de las pérdidas básicas de propagación que requieren el conocimiento del perfil del terreno entre el transmisor y el receptor y resultan idóneos para enlaces punto a punto. Cuando el terreno es orográficamente muy irregular o es de tipo urbano, resulta bastante difícil la modelización de los obstáculos. Para estos casos se han desarrollado numerosos procedimientos

empíricos de estimación de las pérdidas básicas de propagación y de la intensidad de campo, muchos de ellos de aplicación para comunicaciones móviles. Veremos dos métodos muy utilizados y conocidos:

6.1.1 Método de Okumura-Hata

Este método se compone de una serie de curvas que proporcionan valores de la intensidad de campo, para medio urbano, diferentes alturas efectivas de antenas, bandas de 150 MHz, 450 MHz y 900 MHz y una PRA de 1 KW. La altura de la antena de recepción es de 1,5 m, valor típico en aplicaciones móviles.

A estas curvas se les hace una serie de correcciones para tener en cuenta los efectos de ondulación de terreno, pendiente de terreno, presencia de obstáculos significativos, heterogeneidad del terreno (trayectos mixtos tierra/mar), altura de antena receptora, potencia radiada aparente y orientación de calles y densidad de edificación, en caso de zonas urbanas.

Se desarrollaron unas expresiones numéricas para informatizar el método. La fórmula fundamental de Hata, que da las pérdidas básicas de propagación para un medio urbano, es la siguiente:

$$L_b = 69.55 + 26.16 \log f - 13.82 \log h_t - a(h_m) + (44.9 - 6.55 \log h_t) \cdot \log d \quad \text{dB}$$

donde

f : Frecuencia, MHz, en la gama $150 \leq f \leq 1.500$ MHz.

h_t: Altura efectiva de la antena transmisora (m), en la gama $30 \leq h_t \leq 200$ m.

h_m: Altura sobre el suelo de la antena receptora (m) en la gama $1 \leq h_m \leq 10$ m.

d: Distancia (Km) en la gama $1 \leq d \leq 20$ Km.

$a(h_m)$: Corrección que depende de la altura del móvil y del tipo de ciudad.

Si el receptor se encuentra en una zona suburbana, caracterizada por edificaciones de baja altura y calles relativamente anchas, la atenuación es:

$$L_{b_s} = L_b - 2[\log(f/28)]^2 - 5.4$$

Por último, si el receptor se encuentra en una zona rural, abierta, sin obstrucciones en su entorno inmediato, se tiene:

$$L_{b_r} = L_b - 4.78(\log f)^2 + 18.33 \log f - 40.94$$

Hacer notar que la fórmula de Hata no tiene en cuenta la influencia de ondulación del terreno ni los efectos derivados del grado de urbanización.

6.1.2 Método COST 231

Para la predicción más precisa de la pérdida básica de propagación en un medio urbano, se han propuesto varios métodos que incorporan el efecto de las estructuras urbanas (edificios, calles), en cuyo entorno está situado el móvil. Se trata de métodos aplicables a radiocomunicaciones circunscritas totalmente al medio urbano y, en particular, a las comunicaciones móviles celulares, cuando se desea delimitar con precisión razonable la cobertura de la estación transmisora que configura la celda.

Este método es aplicable en situaciones de propagación para las cuales el rayo directo entre el transmisor y el receptor está obstruido por los edificios; algunos de los parámetros utilizados son la altura sobre el suelo de la antena de la estación base, la altura sobre el suelo de la antena el móvil, la altura media de los edificios, la anchura de la calle donde se encuentra el móvil, la anchura entre centros de edificios, el ángulo de inclinación del rayo, etc.

De acuerdo con el método, la pérdida básica de propagación se define como:

$$L_b = L_{bf} + L_{rts} + L_{msd}$$

donde:

L_{bf} es la pérdida en condiciones de espacio libre.

L_{rts} es la pérdida debida a la difracción terraza-calle entre la terraza de los edificio y el móvil.

L_{msd} es una estimación de la difracción multiobstáculo que experimenta el rayo entre la antena transmisora y el edificio próximo al receptor, debida a los edificios interpuestos entre ambos.

6.2 Aplicación

En este capítulo vamos a tratar de, a partir de una serie de medidas de nivel de señal (potencia) correspondiente a una celda de telefonía móvil en una determinada región, predecir el nivel de señal en cualquier punto de esa región. Para ello emplearemos una red neuronal que dispondrá de dos nodos de

entrada, que se corresponderán con las coordenadas de los puntos en cuestión dentro de esa región (coordenada x, coordenada y) y un nodo de salida, que no será más que el nivel se señal ó potencia recibido en ese punto procedente de la celda en estudio. Para realizar esta aplicación dispusimos de valores para 78 puntos dentro de una determinada región. Para poder entrenar una red que realice una predicción de los valores de potencia, dividimos esas 78 muestras en dos grupos, un grupo de 39 muestras que emplearemos para entrenar la red neuronal, y otro grupo de otras 39 muestras que emplearemos para comprobar el funcionamiento de la red neuronal cuando le introducimos como valores de entrada muestras no empleadas en el entrenamiento, es decir, para comprobar si la red neuronal nos sirve para predecir el nivel de señal recibido en una determinada región. A continuación se muestran unas tablas con los valores utilizados en el entrenamiento y para testear la red neuronal.

- Datos para el entrenamiento:

Coordenada X	**Coordenada Y**	**Nivel de potencia recibida (dbm)**
-5.98393333	37.37316667	-90
-5.98298333	37.3757	-81
-5.98296667	37.37575	-80
-5.98291667	37.3755	-92
-5.98291667	37.37873333	-68
-5.98291667	37.3788	-68
-5.98288333	37.37518333	-96
-5.98285	37.3747	-92
-5.98285	37.37918333	-65
-5.98281667	37.37616667	-80
-5.98281667	37.37918333	-67
-5.98268333	37.37305	-88
-5.98268333	37.37948333	-65
-5.98253333	37.3735	-95
-5.98251667	37.3738	-93
-5.98251667	37.3744	-96
-5.98251667	37.38061667	-64
-5.98246667	37.37968333	-60
-5.9824	37.38055	-65
-5.98238333	37.37341667	-104
-5.98233333	37.38061667	-62
-5.98231667	37.37985	-63
-5.98223333	37.38023333	-70

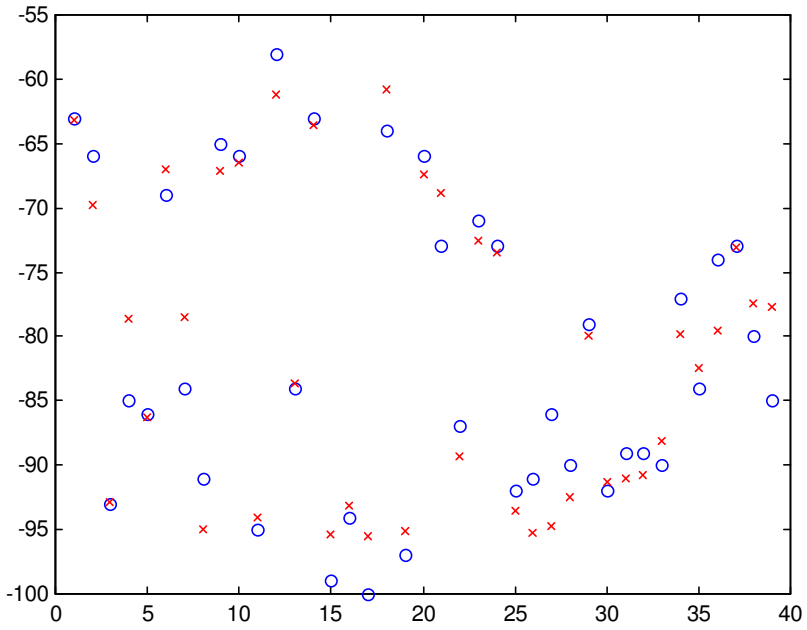
-5.98183333	37.37611667	-93
-5.98153333	37.3808	-79
-5.98146667	37.37603333	-96
-5.98108333	37.37565	-91
-5.98066667	37.37656667	-92
-5.9801	37.37648333	-93
-5.98006667	37.3806	-79
-5.97993333	37.37566667	-88
-5.97986667	37.37626667	-86
-5.97983333	37.37593333	-91
-5.9794	37.3754	-89
-5.97861667	37.38003333	-80
-5.97831667	37.37513333	-76
-5.97795	37.37785	-72
-5.97785	37.37675	-76
-5.97771667	37.3761	-81

- Datos para testear la red:

Coordenada X	**Coordenada Y**	**Nivel de potencia recibida (dbm)**
-5.98243333	37.3806	-63
-5.9822	37.3804	-66
-5.98338333	37.37315	-93
-5.983	37.37586667	-85
-5.98295	37.3756	-86
-5.98295	37.37871667	-69
-5.98293333	37.37596667	-84
-5.98286667	37.37495	-91
-5.98283333	37.37876667	-65
-5.9827	37.37881667	-66
-5.98266667	37.37455	-95
-5.98256667	37.37965	-58
-5.98248333	37.37611667	-84
-5.98248333	37.38061667	-63
-5.98246667	37.37351667	-99
-5.98246667	37.37415	-94

-5.98241667	37.37346667	-100
-5.9824	37.3797	-64
-5.98233333	37.37331667	-97
-5.98225	37.38005	-66
-5.98221667	37.38053333	-73
-5.98215	37.37608333	-87
-5.98211667	37.38055	-71
-5.98203333	37.38083333	-73
-5.98103333	37.37565	-92
-5.98096667	37.37613333	-91
-5.98085	37.37621667	-86
-5.98078333	37.3765	-90
-5.98078333	37.38068333	-79
-5.98048333	37.37661667	-92
-5.98005	37.37568333	-89
-5.98003333	37.3757	-89
-5.97981667	37.37636667	-90
-5.97931667	37.38026667	-77
-5.9789	37.37521667	-84
-5.97793333	37.37985	-74
-5.9778	37.37738333	-73
-5.97775	37.37516667	-80
-5.9776	37.37541667	-85

Para poder trabajar con la red neuronal, hemos normalizado estos datos previamente para tenerlos siempre en el rango de 0 a 1. Con una sola capa oculta, un número determinado de neuronas y suficiente entrenamiento se puede conseguir aproximar a la perfección el conjunto de datos de entrada. El problema es que la red resultante aproxime correctamente datos no usados en el entrenamiento. Después de muchas pruebas, con la red con la que hemos conseguido un mejor resultado al presentarle datos no utilizados en el entrenamiento es una red de 2 capas, con 10 y 5 neuronas respectivamente, totalmente conectada. A continuación se muestra una gráfica en la que aparecen representados los valores de potencia para las distintas muestras del conjunto de test (círculos azules) y los valores (predicciones) obtenidos por la red neuronal:



Como se puede observar, se han obtenido unos resultados bastante aceptables, teniendo en cuenta las diferentes variables que influyen en la toma de muestras, tales como la variabilidad de la señal recibida en un mismo punto, la precisión en la toma de coordenadas de los puntos en los que se ha tomado las muestras, etc.

7 APLICACIÓN DE LAS REDES NEURONALES PARA LA COMPRESIÓN DE IMÁGENES DE ESCALA DE GRISES

7.1 Fundamentos

Las técnicas de compresión de imagen explotan la redundancia que existe naturalmente en la mayoría de las imágenes para un almacenamiento eficiente y/o para la transmisión de éstas a través de algún medio. Aquí, una imagen es codificada con un menor número de bits que el número total de bits requerido para describirla exactamente. La imagen codificada o "comprimida" deberá ser decodificada más tarde en una imagen completa. La compresión de imágenes puede ser entendida como un problema de optimización, donde, idealmente, la codificación y la decodificación son realizadas de forma que se optimice la calidad de la imagen decodificada.

Considérese una red neuronal de tipo feedforward con una única capa oculta. Esta red posee el mismo número de unidades en la capa de salida que entradas tiene esta capa, y el número de unidades en la capa oculta se asume que es mucho menor que el vector de entrada. Se asume que las unidades ocultas poseen

una función de activación del tipo tangente hiperbólica y las unidades de salida son de tipo lineal. Esta red es entrenada mediante un conjunto de vectores de dimensión n de valores reales de forma que a la salida se tenga el mismo vector, es decir, de modo autoasociativo. Por tanto, la red es entrenada para que actúe como un codificador de patrones de valores reales. Se puede decir que la primera parte de la red neuronal, compuesta por los nodos de entrada y las neuronas de la capa intermedia, será la encargada del proceso de compresión, cuyo resultado serán los valores a la salida de las neuronas de la capa oculta; mientras, las neuronas de la capa de salida serán las encargadas de realizar el proceso de descompresión, ya que a partir de la imagen comprimida, se encargará de restaurar los valores de la imagen original.

Supongamos que una red de este tipo recibe entradas de una región de 8x8 pixeles (n=64) y tiene 16 unidades ocultas. Se usa el algoritmo Backpropagation para entrenar la red para autoasociar partes de una imagen seleccionadas de una manera aleatoria. Después del entrenamiento, la red es usada para comprimir y luego reconstruir la imagen, pedazo a pedazo, usando un conjunto de pedazos no solapados que cubran toda la imagen. Entonces, para almacenar la imagen, solo se necesitan 16 valores para cada pedazo de 8x8 de la imagen original, con lo que se alcanza una compresión de 4 a 1.

Hay que hacer notar que, para conseguir una verdadera compresión para el propósito de una eficiente transmisión a través de un enlace de comunicación digital, las salidas de las neuronas ocultas deben ser cuantizadas. La cuantización consiste en la transformación de la salida de las unidades ocultas (que toman valores entre −1 y +1) a un determinado rango entero correspondiéndose al número de bits requeridos para la transmisión. Esto restringe la información en las salidas de las unidades ocultas al número de bits usados.

¿Cómo funciona esta red autoasociativa sobre nuevas imágenes? Intuitivamente, uno no esperaría que esta red generalizase a partir de una sola imagen de entrenamiento, puesto que la red aprende, en algún sentido, las características estadísticas de la imagen con la que es entrenada, y cada imagen tiene unas características estadísticas diferentes. Sorprendentemente, se puede comprobar que la red se comporta bastante bien al reproducir imágenes con las que no fue entrenada, incluso cuando se utiliza la cuantización. También se puede comprobar que se consigue una mejor reproducción de una imagen cuando se realiza la cuantización de las salidas de las neuronas ocultas durante el proceso de entrenamiento, en lugar de ser usada únicamente para la reproducción.

7.2 Aplicación

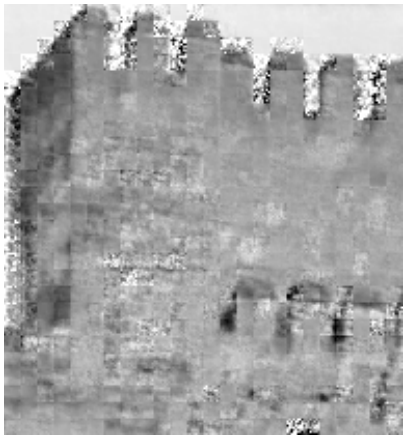
En nuestro caso hemos empleado una red neuronal con 64 nodos de entrada y 64 neuronas a la salida y una capa oculta. La capa oculta tendrá un número menor de neuronas que el número de neuronas a la salida. En esta capa oculta emplearemos la función tangente hiperbólica. La imagen empleada es una imagen de tamaño 192x208 pixels, que dividiremos en regiones de 8x8 pixels (64 valores) no solapantes, que emplearemos para entrenar la red neuronal con el algoritmo Backpropagation y de manera autoasociativa, es decir, entrenando la red neuronal para que a la salida se obtengan los mismos valores

que a la entrada. La imagen empleada es de escala de grises con 8 bits, es decir, un color se representa mediante un número de 1 a 256, con lo que se ha realizado una transformación para ajustarlos entre –1 y +1 a la hora de entrenar la red neuronal. A continuación se muestran los distintos casos estudiados, cada uno empleando un número distinto de neuronas en la capa oculta:

7.2.1 Empleando 32 neuronas en la capa oculta

En este caso, empleando 32 neuronas, conseguimos una compresión del 50 %, ya que una región de la imagen de tamaño 8x8 pixels que se representa con 64 valores, se representará con tan sólo 32 valores. A continuación se muestra la imagen empleada y el resultado obtenido por la red neuronal, es decir, la imagen resultante después de descomprimir la imagen comprimida por la red neuronal:

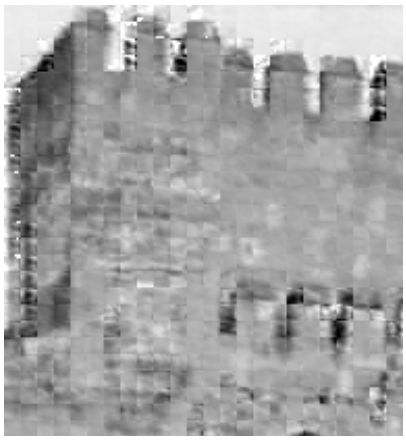
Se puede ver como los peores resultados se obtienen cuando se producen cambios bruscos de tonos. A continuación se muestran los resultados obtenidos cuando se le presenta a la red una imagen totalmente diferente y para la cual no ha sido entrenada:



Se puede ver cómo a pesar de no haber sido entrenada para esta imagen los resultados no desmerecen mucho en comparación con los obtenidos con la primera imagen.

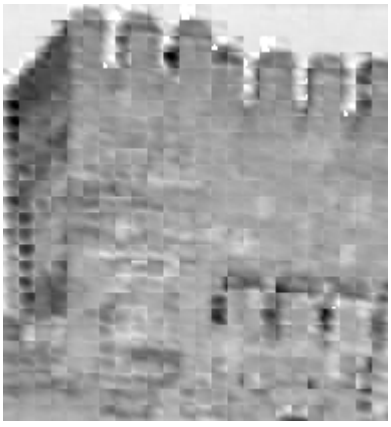
7.2.2 Empleando 16 neuronas en la capa oculta

A continuación se muestran los resultados cuando se usan tan solo 16 neuronas en la capa intermedia, con lo que se conseguiría una compresión del 75 %. Representamos tanto el resultado para la imagen con la que se ha entrenado la red neuronal como la imagen con la que no se ha entrenado:



7.2.3 Empleando 8 neuronas en la capa oculta

Finalmente se muestran los resultados cuando se usan tan solo 8 neuronas en la capa intermedia, con lo que se conseguiría una compresión del 87.5 %. Nuevamente, representamos tanto el resultado para la imagen con la que se ha entrenado la red neuronal como la imagen con la que no se ha entrenado:



Como se puede ver, a medida que disminuye el número de neuronas de la capa intermedia, se obtiene un peor resultado al descomprimir la imagen comprimida por la red neuronal. En el caso de 8 neuronas, se aprecian claramente la división en regiones de 8x8 pixels empleada. Además, se comprueba que la red se

comporta de una manera bastante aceptable cuando se aplica la compresión a una imagen para la que no ha sido entrenada la red neuronal.

Como ya hemos comentado, el objetivo de esta aplicación es la reducción del tamaño que ocupa una imagen para su transmisión mediante algún medio físico o para su almacenamiento. Cuando hemos hablado de la red neuronal de 32 neuronas, hemos dicho que se conseguía una compresión del 50 %. Esto no es realmente así. Para conseguir realmente esa compresión del 50 %, el número de bits utilizados para la representación de los datos de la imagen comprimida deben ser igual al número de bits utilizados para la representación de los datos de entrada. Es decir, si estamos tratando con una imagen de escala de grises de 256 tonos, necesitamos 8 bits para especificar el color de un punto. Si de 64 puntos de 8 bits pasamos a 32 puntos de 16 bits, no estaríamos consiguiendo ninguna compresión. Por tanto, los valores obtenidos para la imagen comprimida deben ser de una longitud de 8 bits, por lo que habría que truncarlos en caso de que fuese necesario. Este fenómeno, denominado cuantización, influye negativamente en los resultados obtenidos, empeorando la calidad de la imagen descomprimida.

8 LÍNEAS DE AVANCE

8.1 Compresión de imágenes

En el apartado anterior, hemos visto cómo las redes neuronales pueden ser utilizadas para comprimir imágenes de escala de grises. Hemos visto como, además, tras entrenar la red neuronal con datos pertenecientes a una imagen en concreto, la red era capaz de comportarse bien cuando le eran presentado imágenes con los que no había sido entrenadas, es lo que se denomina capacidad de generalización, aunque evidentemente, no se puede pretender que haga la compresión con la misma calidad. Así mismo, hemos hablado de un fenómeno, denominado cuantización, que degrada el resultado de la compresión. Se propone hacer un estudio de la incidencia de la cuantización en los resultados obtenidos, así como el diseño y análisis de nuevos métodos de compresión, por ejemplo, modificando el tamaño de los patrones utilizados para entrenar la red neuronal.

Además, hoy en día, en el mundo de la imagen, predomina el color. Se podría pensar en utilizar las redes neuronales para comprimir imágenes en color como una extensión de la aplicación vista anteriormente, de forma que se obtenga un método más general.

8.2 Reconocimiento de dígitos escritos a mano

El problema del reconocimiento de dígitos escritos a mano es un problema clásico de reconocimiento de patrones. Se trata de crear una red neuronal que sea capaz de decir qué número se trata el representado

por una mapa de bits obtenido a partir de muestras escritas. Para este tarea se puede diseñar una red neuronal basada en el algoritmo de propagación hacia atrás (Backpropagation). En la literatura se pueden encontrar diversos métodos para realizar esta aplicación, uno de los cuales vamos a ver a continuación.

Supongamos que tenemos un conjunto de ejemplos. Estos ejemplos son preprocesados a través de una simple transformación lineal que transforma los dígitos en una imagen de tamaño 16 x 16 de niveles de grises. Los niveles de grises de cada imagen son escalados y traducidos para que caigan en el rango de –1 a +1.

La red se compone de tres capas ocultas, que denominaremos H1, H2 y H3, y una capa de salida. La capa H1 está conectada a la imagen de entrada. La capa de salida contiene 10 unidades, y cada una representará un dígito. La capa H3 tiene 30 unidades y está conectada por completo con la capa H2 y con la capa de salida. Para las conexiones entre la capa de entrada y la capa H1 y las conexiones entre la capa H1 y la capa H2, se usa compartición de pesos. Esta compartición de pesos consiste en tener varias conexiones controladas por un único peso, de forma que se reduce el número de parámetros libres de la red, lo que lleva a mejorar la generalización. Otro motivo para emplear la compartición de pesos es para hacer que las capas ocultas H1 y H2 desarrollen propiedades de selección de características que simplifiquen la tarea de clasificación implementada por la capa H3 y la capa de salida. Esta estrategia es fundamental para alcanzar una gran exactitud en la clasificación.

La primera capa oculta (H1) está compuesta por "mapas característicos". Todas las unidades de un mapa característico comparten los mismos pesos (excepto por el umbral, que puede definir de una unidad a otra). Hay 12 grupos de mapas característicos de tamaño 8x8 (64 unidades por mapa característico). Cada unidad en un mapa característico recibe las entradas de una ventana de tamaño 5x5 de la imagen de entrada. Dos unidades vecinas de una mapa característico en H1 tiene su campo receptivo (ventana de tamaño 5x5) desplazado dos pixeles. Cada una de las 64 unidades en un mapa característico realiza la misma operación sobre las partes correspondientes de la imagen. La función realizada por un mapa característico puede ser interpretada como la detección de 1 de 12 posibles características en posiciones arbitrarias de la imagen de entrada.

La capa H2 también está compuesta de 12 mapas característicos, cada uno de tamaño 4x4 (16 unidades). El esquema de conexionado entre las capas H1 y H2 es muy similar al descrito entre la capa H1 y la capa de entrada, pero con algunas complicaciones más debido a los múltiples mapas característicos de tamaño 8x8 de la capa H1. Cada unidad de la capa H2 recibe entradas de un conjunto (8 en este caso) de los 12 mapas de la capa H1. Su campo receptivo está compuesto de una ventana de tamaño 5x5 centrada en idéntica posición en cada uno de los 8 mapas característicos de los que recibe entradas. De nuevo, todas las unidades en un mapa de la capa H2 comparten sus pesos.

Como resultado de esta estructura, la red tiene 1.000 unidades, 64.660 conexiones y 9.760 pesos independientes. Todas las unidades usan la función de activación tangente hiperbólica. Antes de proceder

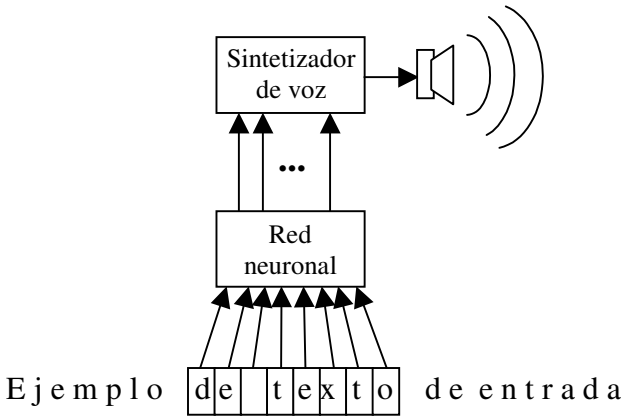
al entrenamiento, los pesos son inicializados con una distribución aleatoria uniforme entre –2.4 y +2.4, llevando después a cabo una normalización dividiendo cada peso por el fan-in de la unidad correspondiente.

Se trataría de realizar un estudio de los porcentajes de clasificación correcta cuando se le presentan a la red ejemplos no vistos anteriormente, y compararlos con otros posibles métodos de reconocimiento de dígitos escritos a mano.

8.3 Sintetizador de texto escrito

Se trataría de realizar una aplicación que fuese capaz de convertir texto escrito en castellano en voz. En la literatura se pueden encontrar diversas aplicaciones realizadas con este fin. A continuación veremos una de ellas, denominada NETtalk, que es capaz de convertir texto en inglés a voz. Consta de dos módulos: una red neuronal encargada del mapeado y un sintetizador de voz comercial. La red neuronal es una red feedforward que posee 80 unidades en su capa oculta y 26 en la capa de salida. Las 26 unidades de la capa de salida forman un código que representarán los fonemas. La salida de la red neuronal dirige el sintetizador, que irá generando los sonidos asociados con los fonemas de entrada. La entrada de la red neuronal es un vector binario de dimensión 203 que codifica una ventana de 7 caracteres consecutivos (29 bits por cada uno de los caracteres, incluyendo puntuación). La salida deseada es un código de un fonema que da la pronunciación de la letra central de la ventana de entrada.

A continuación se muestra un diagrama de bloques de esta aplicación:



Cuando fue entrenada con 1024 palabras de un conjunto de ejemplos de fonemas ingleses, NETtalk era capaz de "leer" después de solo 10 ciclos de entrenamiento, y obtuvo una exactitud del 95 % en el conjunto de entrenamiento después de 50 ciclos. La red aprendió primero a reconocer los puntos de división entre las palabras y después aprendió gradualmente a asociar fonemas, sonando un poco como un niño que está aprendiendo a hablar. La red era capaz de distinguir entre vocales y consonantes, y cuando fue probada con un texto nuevo, alcanzó un grado de generalización del 78 %. Después de añadir ruido

aleatorio a los pesos o quitar varias unidades, se comprobó que el rendimiento de la red se degradó de una manera gradual y no de manera catastrófica, como suele ocurrir en muchos sistemas digitales. Esto es una gran ventaja, ya que si una vez implementado el sistema se estropea algún componente, el sistema funcionará algo pero, pero no dejará de funcionar.

Existe un sistema similar disponible comercialmente denominado DEC-talk que emplea un sistema experto de reglas lingüísticas de escritura que funciona mejor que NETtalk al desempeñar la misma tarea. Sin embargo, lo significativo de NETtalk es el poco tiempo necesitado para su desarrollo; NETtalk simplemente aprendió de un conjunto limitado de ejemplos, mientras que DEC-talk incluye reglas que son el resultado de varios años de análisis por multitud de lingüísticos. Esta aplicación muestra como se puede utilizar un sistema basado en redes neuronales para solventar problemas que no son bien conocidos.

9 BIBLIOGRAFÍA

- Simon Haykin. 1994. "Neural Networks. A comprehensive foundation"
- Mohamad H. Hassoun. 1995. "Fundamentals of Artificial Neural Networks"
- José María Hernando Rábanos, 1995. "Transmisión por radio"
- Iltcho Angelov, Herbert Zirath, and Niklas Rorsman. "A New Empirical Nonlinear Model for HEMT and MESFET Devices". IEEE Transactions on Microwave Theory and Techniques VOL. 40 NO 12. December 1992.
- Fujii. "Large-signal Switching MESFET Model for IMD Analysis". IEEE Transactions on Microwave Theory and Techniques VOL. 48 NO 3. March 2000.
- http://carol.wins.uva.nl/~portegie/matlab/nnt. "Introduction to the Matlab Neural Network Toolbox 3.0".