Introducción a las Redes Neuronales Artificiales.

Ing. Mario Alberto Ibarra-Manzano.

Marzo del 2006.

Resumen

LA teoría y el diseño de las Redes Neuronales Artificiales (RNA) han tenido un avance significativo durante los últimos 20 años. Mucho de este progreso está ligado directamente con el procesamiento de señales. En particular, la naturaleza no lineal y la capacidad de las RNA para aprender de su entorno de manera supervisada y no supervisada, así como su propiedad de ser un aproximador universal, hacen de ellas una posible solución muy buena para resolver problema difíciles en el procesamiento de señales y especialmente en el área del reconocimiento de patrones.

Desde la perspectiva del procesamiento de señales, es indispensable desarrollar una compresión apropiada de las estructuras básicas de las RNA y como éstas impactan en los algoritmos y aplicaciones del procesamiento de señales. Un problema en el diseño de las RNA consiste en identificar aquellas estructuras neuronales que se pueden utilizar para resolver problemas reales, de aquellas que todavía están bajo desarrollo o tienen dificultades que resaltaran al momento de aplicarse en un problema real. Al tratar con aplicaciones de procesamiento de señales, es crítico entender la naturaleza del problema para poder así aplicar la estructura más apropiada en cada caso. Además, también es importante evaluar el desempeño, en las RNA, su robustez y la relación velocidad-efectividad en el sistema de procesamiento de señales, así como también desarrollar las metodologías para integrar las redes neuronales con otros algoritmos del procesamiento de señales. Otro problema importarte es como evaluar los modelos en las redes neuronales, algoritmos de entrenamiento y estructuras e identificar aquéllas que resuelven los problemas de manera fiable de las que no.

Índice general

Capítulo 1

Introducción a las Redes Neuronales Artificiales.

1.1. ¿Qué es un Red Neuronal Artificial?

El trabajo con RNA, ha sido motivado desde su inicio por el conocimiento de que el cerebro calcula de una manera enteramente diferente a como lo hacen las computadoras digitales. El cerebro está constituido por estructuras llamadas neuronas. Las neuronas típicamente son 5 ó 6 veces más lentas que las compuertas lógicas de silicio; los eventos en un chip de silicio ocurren en el rango de nanosegundos, mientras que los eventos en las neuronas ocurren en milisegundos. Sin embargo, el cerebro está estructurado para una velocidad de operación relativamente baja de una neurona debido a que tiene un número verdaderamente inmenso de neuronas con interconexiones masivas entre ellas, es estimado que debe ser del orden de 10 billones de neuronas con, aproximadamente, 60 trillones de sinapsis o interconexiones en la corteza del hombre. La red que resulta es el cerebro el cual es una estructura enormemente eficiente. Específicamente, la eficiencia energética del cerebro es aproximadamente de 10^{-16} Joules por operación por segundo, mientras que el valor correspondiente para las mejores computadoras de hoy es aproximadamente de 10^{-6} Joules por operación por segundo.

El cerebro es una computadora altamente compleja, no lineal y paralela (sistema de información-procesamiento). Éste tiene la capacidad de organizar neuronas así como desempeñar ciertos cálculos (reconocimiento de patrones, percepción y control de motores) muchas veces más rápido que la computadora digital más rápida que existe en la actualidad. Considere, por ejemplo, la visión humana, la cual es una tarea de información-

Figura 1.1: Célula neuronal biológica.

procesamiento. La función del sistema visual es proveer una representación del ambiente a nuestro alrededor y, más importante, proporcionar información que necesitamos para interactuar con el ambiente. Para ser específicos, el cerebro rutinariamente alcanza la tarea de reconocimiento perceptual (es decir, reconocer una cara familiar en una escena no familiar) en el orden de 100 a 200 ms, mientras que una tarea de mucha menor complejidad tomará días para que una computadora convencional pueda realizarla.

Así como las neuronas biológicas son esenciales como unidades de procesamiento de información en el funcionamiento del cerebro humano, de la misma manera las RNA han arreglado sus neuronas artificiales. En la forma más general, una neurona es una máquina diseñada para modelar la manera en la que el cerebro desempeña una tarea particular o función de interés; la red es usualmente implementada usando componentes electrónicos o simulados en software sobre una computadora digital. El principal interés en el trabajo sobre RNA es encerrar enteramente una importante clase de redes neuronales que desempeñen cálculos útiles a través del proceso de aprendizaje. Para alcanzar este buen desempeño, las RNA emplean una interconexión masiva de células computarizadas simples referidas como neuronas o unidades de procesamiento. Una RNA vista como una máquina adaptiva se puede definir como:

Un procesador distribuido paralelo que tiene una tendencia natural para almacenar conocimiento en base a la experiencia y hacerlo disponible para su uso. Ésta imita al cerebro en dos aspectos:

1. El conocimiento es adquirido por la red en base a un proceso de aprendizaje.
2. Las longitudes de las conexiones interneuronas conocida como pesos sinápticos son usados para almacenar el conocimiento.

El procedimiento usado para desempañar el proceso de aprendizaje es llamado algoritmo de aprendizaje, la función en la cual son modificados los pesos sinápticos de la red de manera ordenada para alcanzar un objetivo deseado.

La modificación de los pesos sinápticos provee el método tradicional para el diseño de RNA. Tal aproximación es cercana a la teoría de filtros lineales adoptivos, la cual ya está bien establecida y es exitosamente aplicada en diversos campos tales como las comunicaciones, control, radar, sonar, sismología e ingeniería biomédica. Sin embargo, además es posible para una RNA modificar su propia topología, lo cual es motivado por el hecho de que las neuronas en el cerebro humano mueren y que nuevas conexiones sinápticas crecen.

1.2. Beneficios de las RNA.

En una RNA su poder de cómputo se deriva de dos razones principales, primero de su estructura distribuida totalmente paralela y segundo de su habilidad para aprender y por tanto generalizar; la generalización se refiere a que la red neuronal produzca salidas razonables para entradas no encontradas durante su entrenamiento (aprendizaje). Estas dos capacidades de procesamiento de información hacen posible que las RNA resuelvan problemas complejos (gran escala) que son realmente no tratables. Sin embargo, en la práctica las RNA por si solas no pueden proveer la solución. Por lo tanto, necesitan ser integradas en un sistema consistente. Específicamente, un problema complejo es descompuesto en un número de tareas relativamente simples, y las RNA son asignadas a un subconjunto de las tareas (por ejemplo reconocimiento de patrones, memoria asociativa, control) eso si ajusta a sus capacidades inherentes. Es importante reconocer que aun hay un largo camino por recorrer antes de poder construir (si se puede) una arquitectura que imite al cerebro humano.

El uso de las RNA ofrece las siguientes capacidades y propiedades útiles:

1. **No linealidad.** Una neurona es básicamente un dispositivo no lineal. Por lo tanto, una RNA, hecha de una interconexión de neuronas, es por si misma no lineal. Más aún, la no linealidad es una propiedad especial en el sentido de que eso está distribuido a través de la red.

No linealidad es una propiedad altamente importante, en particular si el mecanismo físico fundamental responsable de la generación de una señal de entrada es inherentemente no lineal.

2. **Mapeo de entrada-salida.** El aprendizaje supervisado es la modificación de los pesos sinápticos de una RNA aplicando un conjunto de muestras de entrenamiento o ejemplos. Cada ejemplo consiste de una única señal de entrada y su correspondiente salida deseada. Se evalúa la RNA con un ejemplo, el cual es tomado de forma aleatoria de un conjunto. Los pesos sinápticos de la red son modificados para minimizar la diferencia entre la respuesta deseada y la respuesta real de la red producida debido a la señal de entrada, tomando como base un criterio estadístico apropiado. El entrenamiento de la red es repetido para muchos ejemplos en el conjunto hasta que ésta alcanza un estado estable, en el cual ya no hay cambios significativos en sus pesos sinápticos; los ejemplos de entrenamiento previamente aplicados pueden ser reaplicados durante la sesión de entrenamiento pero en un orden diferente. Así la red aprende de los ejemplos construyendo un mapeo de entrada-salida para el problema.
3. **Adaptividad.** Las RNA tienen la capacidad de adaptar sus pesos sinápticos a los cambios del ambiente que los rodea. En particular, una RNA entrenada para operar en un ambiente específico puede ser fácilmente reentrenada para tratar con cambios menores en las condiciones de su ambiente de operación. Más aún, cuando ésta opera en un ambiente no estacionario (esto es un ambiente cuyas estadísticas cambian con el tiempo), una RNA puede ser entrenada para cambiar sus pesos sinápticos en tiempo real. La arquitectura natural de una RNA para clasificación de patrones, procesamiento de señales y aplicaciones de control, junto con la capacidad adaptiva de la red, la hacen una herramienta ideal para su uso en la clasificación de patrones adaptivos, procesamiento de señales adaptivas y control adaptivo.
4. **Respuesta evidencial.** En el caso de la clasificación de patrones, una RNA puede ser diseñada para proveer información no solamente de un patrón particular seleccionado, sino además de la confidencia en la decisión hecha. Esta última información puede ser usada para rechazar patrones ambiguos que podrían aparecer, y por tanto incrementar el desempeño de clasificación de la red.
5. **Información contextual.** El reconocimiento de un patrón es representado por el estado de activación y estructural de la RNA. Cada neurona en la red es potencialmente afectada por las actividades globales de todas las otras neuronas en la red. Por lo tanto, la informa-

ción contextual es tratada naturalmente por una RNA.

6. **Tolerancia a la falla.** Una RNA, implementada en un hardware, es tolerante al error en el sentido que su desempeño bajo condiciones de operación adversas. Por ejemplo, si una neurona o sus conexiones son dañadas, el patrón almacenado en la red es degradado. Sin embargo, debido a la naturaleza de la distribución de la información en la red, el daño no es total, si no es visto como una pérdida de información.
7. **Implementación VLSI.** La naturaleza paralela de la RNA la hace potencialmente rápida para los cálculos de ciertas tareas. Esta misma característica hace a una RNA idealmente apropiada para su implementación usando tecnología de integración a gran escala (VLSI). La característica principal del VLSI es que en él se pueden implementar algoritmos de mucha complejidad en un estilo altamente jerárquico, el cual hace posible el uso de las RNA como una herramienta para aplicaciones en tiempo real en el reconocimiento de patrones, procesamiento de señal y control.
8. **Uniformidad de análisis y diseño.** Las RNA son siempre usadas como procesadores de información, esto es debido a su naturaleza y al hecho de que la misma notación es utilizada en las diversas aplicaciones de éstas. Las neuronas son los elementos comunes de las diversas estructuras en las RNA y sus aplicaciones. Redes modulares pueden ser construidas a través de una integración de módulos comunes mediante modificaciones menores en las estructuras.
9. **Analogía neurobiológica.** El diseño de las RNA es motivada por su analogía con el cerebro humano, el cual es una prueba viviente de que el procesamiento paralelo tolerante al error no sólo es físicamente posible sino que además rápido y poderoso. Los neurologistas ven las RNA como una herramienta de investigación para la interpretación de los fenómenos neurobiológicos. Por otro lado los ingenieros miran la neurobiología en busca de nuevas ideas para resolver problemas más complejos que los basados en técnicas de diseño convencionales.

1.3. Aplicaciones de las RNA.

Las RNA tiene muchas aplicaciones entre ellas sobresalen las siguientes:

- **Empresariales:** Un estudio de RNA realizado por DARPA en 1988, enumera varias aplicaciones de las RNA. Este estudio comienza en 1984 con el ecualizador de canal adaptativo. Este dispositivo, el cual es un éxito comercial sobresaliente, es una RNA simple usada en siste-

mas de telefonía de larga distancia para estabilizar las señales de voz. El reporte del DARPA enumera otras aplicaciones comerciales, incluyendo un reconocedor pequeño de palabras, un monitor de proceso, un clasificador de sonar y un sistema de análisis de riesgos. Las RNA han sido aplicadas en muchos otros campos desde que el reporte de DARPA fue publicado.

- **Aeroespaciales:** En el campo aeroespacial sobresalen las siguientes aplicaciones: la simulación de ruta de vuelo, los sistemas de control de aviones, los pilotos automáticos de alto desempeño, la simulación de partes aeroespaciales y la detección de fallas en componentes de un avión.
- **Automotrices:** En el área automotriz las aplicaciones son: sistemas guía automáticos y el análisis de actividad seguras.
- **Bancarias:** Las aplicaciones bancarias son: la lectura de cheques y otros documentos, la evaluación de aplicaciones de crédito. Además las RNA son usadas para el reconocimiento de actividades inusuales en tarjetas de crédito, que posiblemente podrían estar asociadas con la pérdida o robo de una tarjeta de crédito.
- **Militares:** Las principales aplicaciones en la milicia son: la guía de armas, el seguimiento de objetivos, discriminación de objetos, reconocimiento de rostros, nuevos tipos de sensores, sonares, radares y procesamiento de señales en imágenes incluyendo compresión de datos, extracción de características y supresión de ruido.
- **Electrónicas:** En la electrónica las RNA se utilizan para la predicción en la secuencia de códigos, diseño de circuitos integrados, control de procesos, análisis de fallas en circuitos y chips, visión artificial, síntesis de voz y modelado no lineal.
- **Entretenimiento:** La animación, los efectos especiales y el pronóstico de ventas son algunas de las aplicaciones de las RNA.
- **Finanzas:** La principales aplicaciones financieras son: la valoración del estado real, asesor de préstamos, análisis financiero corporativo, predicción de precio actual, análisis del crédito en línea entre otras.
- **Industriales:** Las RNA están siendo entrenadas para predecir la salida de gases en hornos y otros procesos industriales. Las RNA sustituyen equipos complejos y costosos empleados para este propósito.
- **Aseguradoras:** En el área de las aseguradoras la aplicaciones son: evaluación para aplicación de pólizas y optimización de productos.
- **Manufactura:** En la manufactura las aplicaciones son: el control de procesos de manufactura, análisis y diseño de productos, diagnóstico de maquinaria y procesos, identificación de partículas en tiempo real,

análisis de calidad en soldaduras y en chips de computadoras, análisis de mantenimiento de maquinas, análisis en el diseño de productos químicos, modelado dinámico de procesos químicos.

- **Médicas:** Algunas aplicaciones médicas son: el análisis de células para la detección del cáncer de pecho, análisis de ECG y EEG, diseño de prótesis, optimización en el tiempo de trasplantes, incremento en la calidad de los hospitales, reducción en los gastos de hospital, pruebas de consideración en la sala de emergencias.
- **Robóticas:** El control de trayectorias, robot elevador, controlador manipulador y sistemas de visión son algunas de las aplicaciones de las RNA en la robótica.
- **Habla:** En el procesamiento del habla las principales aplicaciones de las RNA son: el reconocimiento del habla, la compresión de la señal del habla, la clasificación de vocales y la síntesis del habla.
- **Mercadeo:** En el mercadeo sobresalen las siguientes aplicaciones: el análisis de mercado y sistemas de advertencia del mercado bursátil.
- **Telecomunicaciones:** Las Telecomunicaciones utilizan las RNA para la compresión de imágenes y datos, en los servicios de información autómata, en la traducción del lenguaje hablado en tiempo real, para sistemas de procesamiento de pago de usuarios.
- **Transporte:** Los sistemas de diagnóstico de frenos en vehículos de carga y los sistemas de planeación de rutas son algunas de las aplicaciones de las RNA en el transporte.

Capítulo 2

Conceptos fundamentales en Redes Neuronales Artificiales.

2.1. Modelo de un neurona artificial.

Una RNA, de manera general, es un modelo matemático computacional que simula el funcionamiento del sistema neuronal biológico. En 1943, McCulloch, un neurobiólogo, y Pitts, un estadístico, publicaron un artículo titulado "A logical calculus of ideas imminent in nervous activity" en *Bulletin of Mathematical Biophysics*. Este artículo inspiró el desarrollo de la computadora digital moderna, o el cerebro electrónico, como John von Neumann la llamó. Por el mismo tiempo, Frank Rosenblatt motivado también por este artículo investigó el procesamiento del ojo a lo que en el futuro llevó a la primera generación de RNA, conocida como el perceptrón.

Las neuronas son los elementos más pequeños que constituyen la mayoría de las estructuras de RNA. El modelo de la neurona más utilizado es el que se basa en el trabajo de McCulloch y Pitts, el cual es mostrado en la figura 2.1.

Cada neurona está constituida por dos partes: la función de red y la función de activación. La función de red determina el cómo los patrones de entrada $\{x_j; 1 \leq j \leq N\}$ son combinados dentro de la neurona. En la figura 2.1, se utiliza una combinación lineal de los patrones la cual es representada por la ecuación 2.1. En esta ecuación, $\{w_j; 1 \leq j \leq N\}$ son nombrados los pesos sinápticos y en ellos radica el conocimiento aprendido por la red. El parámetro θ es llamado umbral (o sesgo), éste es un peso sináptico interno a

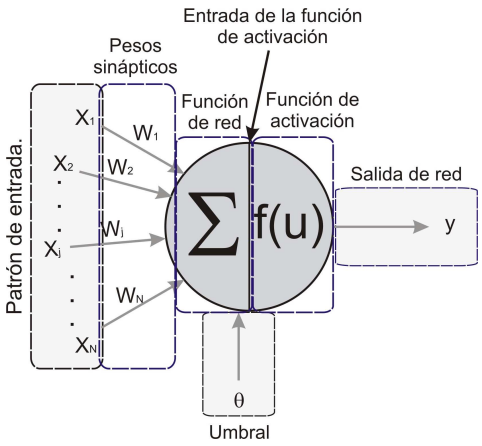


Figura 2.1: Modelo de la neurona de McCulloch y Pitts

Cuadro 2.1: Funciones de red utilizadas en RNA.

Función de red	Modelo matemático
Lineal	$u = \sum_{j=1}^{N} w_j x_j + \theta$
Alto orden (Segundo orden)	$u = \sum_{j=1}^{N} \sum_{k=1}^{N} w_{jk} x_j x_k + \theta$
Delta o Sigma-Pi	$u = \prod_{j=1}^{N} w_j x_j$

la neurona. En la tabla 2.1 son mostradas alguna funciones de red utilizadas en la mayoría de la estructuras neuronales.

$$u = \sum_{j=1}^{N} w_j x_j + \theta \tag{2.1}$$

La salida de una neurona denotada por y en la figura 2.1, está relacionada con la salida de la función de red u, mediante la transformación lineal o no lineal llamada función de activación:

$$y = f(u) \tag{2.2}$$

Las funciones de activación más utilizadas en los diferentes modelos de RNA son mostradas en la tabla 2.2. En esta tabla se muestra los modelos matemáticos y derivadas de las diversas funciones de activación en base a parámetros conocidos. Para una mejor referencia sobre las diferentes funciones de activación las gráficas de éstas son mostradas en el apéndice A.

Cuadro 2.2: Funciones de red de activación utilizada en RNA.

Función de activación	Modelo matemático	Derivada $\left(\frac{df(u)}{u}\right)$
Sigmoidal	$f(u) = \frac{1}{1+e^{-u}}$	$f(u)[1 - f(u)]$
Tangente hiperbólica	$f(u) = \tanh(u)$	$1 - [f(u)]^2$
Tangente inversa	$f(u) = \frac{2}{\pi}\tan^{-1}(u)$	$\frac{2}{\pi[1+u^2]}$
Escalón	$f(u) = \begin{cases} 1 & u > 0; \\ 0 & u < 0. \end{cases}$	La deriva no existe en $u = 0$
Gaussiana	$f(u) = exp\left(-\frac{\|\|u-m\|\|^2}{\sigma^2}\right)$	$-\frac{2(u-m)f(u)}{\sigma^2}$
Lineal	$f(u) = au + b$	a

2.2. Modelo de una RNA.

En una RNA, multiples neuronas son interconectadas para forma una red y distribuir el procesamiento de los datos. La configuración de las interconecciones pueden describirse de manera efectiva mediante una gráfica directa (fig. 2.2). La gráfica consiste de nodos (en el caso de las RNA, neuronas) y conexiones (en el caso de las RNA, pesos sinápticos).

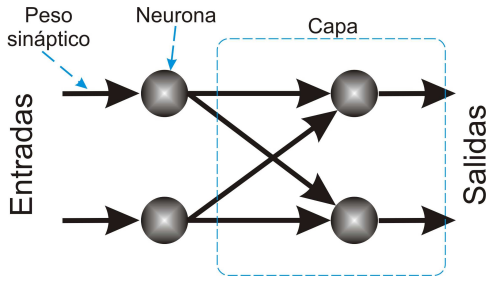


Figura 2.2: Representación gráfica de una RNA.

Las RNA vistas desde su topología pueden clasificarse como cíclica y acíclica. La figura 2.3(a) muestra una RNA con topología acíclica, en ésta no existen lazos de retroalimentación entre capas. Este tipo de topología es empleada para realizar un mapeo no lineal entre las entradas y las salidas. En la figura 2.3(b) se muestra una RNA cíclica, en ésta existe al menos un peso sináptico que realiza una retroalimentación entre capas. Este tipo de RNA son conocidas como redes recurrentes. Debido a los lazos de retro-

alimentación entre capas, las redes recurrentes dan resultados bueno en el modelado de sistemas dinámicos no lineales los cuales requieren de una memoria interna. Sin embargo, las RNA recurrentes frecuentemente presentan un comportamiento complejo que aún es un tópico de investigación en el campo de las RNA.

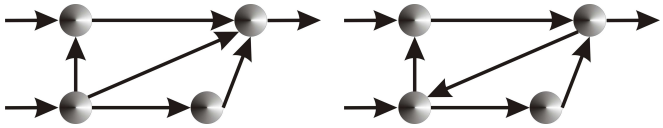


Figura 2.3: Gráfica de (a) una red acíclica. (b) una red cíclica.

2.3. Taxonomía de las RNA.

Las aplicaciones de las RNA tienen dos fases importantes: la fase de aprendizaje [1] o entrenamiento y la fase de prueba. La fase de entrenamiento utiliza un conjunto de datos o patrones de entrenamiento para determinar los pesos sinápticos que definen el modelo neuronal. Una vez entrenado este modelo, es decir, una vez ajustados los pesos para la aplicación asignada a la RNA, se continuará con la fase de prueba o funcionamiento directo, en la que se procesan los patrones de prueba que constituyen la entrada habitual de la RNA, analizándose el funcionamiento correcto de la RNA.

Las aplicaciones reales de las RNA deben resolver dos tipos diferentes de requisitos en el procesamiento. En un caso, se necesita la fase de prueba en tiempo real pero la fase de entrenamiento se realiza fuera de línea. Sin embargo, en algunos casos es necesario que los dos procesos, el de prueba y el de entrenamiento se lleven en tiempo real. Estos dos enfoques requieren de velocidades de procesos muy diferentes, que afectan a los algoritmos y el hardware empleado. Desde el punto de vista del tipo de entrenamiento, una posible taxonomía de las RNA es la mostrada en la figura 2.4.

[1] Una característica de las RNA es su capacidad de aprender. La fase de aprendizaje consiste en la actualización o cambio de los pesos sinápticos que caracterizan a las conexiones. Los pesos son adaptados de acuerdo a la información extraída de los patrones de entrenamiento que se van presentando. Normalmente, los pesos son calculados optimizando (minimizando o maximizando) alguna "función de energía". Por ejemplo, el entrenamiento supervisado minimiza el error cuadrático medio entre el valor deseado y el valor obtenido.

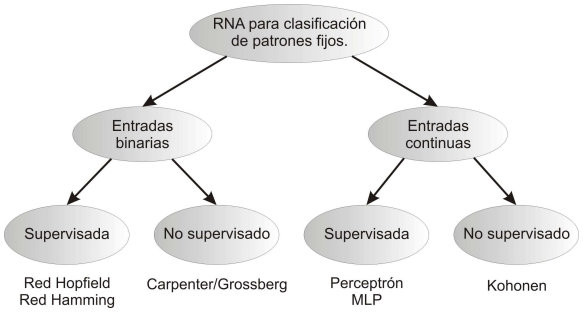


Figura 2.4: Taxonomía de las RNA para clasificación de patrones estáticos.

En la figura 2.4 se muestran 6 importantes modelos de RNA que se puede utilizar en la clasificación de patrones estáticos. La taxonomía presentada es dividida primero entre redes con entradas de valores binarios y continuos. Debajo de éstas, las redes son divididas en redes con entrenamiento supervisado y no supervisado. Las RNA con entrenamiento supervisado [2] tales como la red Hopfield y el perceptrón son usadas como memorias asociativas o como clasificadores. El entrenamiento de RNA sin supervisión, tales como los mapas de características de Kohonen, son utilizados como vectores cuantizados o para formar clusters. Generalmente estos últimos modelos son utilizados cuando no se tiene la información concerniente a la clase correcta de las entradas durante el proceso de entrenamiento. La mayoría de los modelos neuronales tienen un entrenamiento adaptativo, aunque en ciertas ocasiones la red de Hopfield y la de Hamming son usadas con pesos fijos. Entre los modelos mostrados resalta el perceptrón, el cual puede procesar entradas binarias y continuas, frecuentemente este modelo es empleado como un clasificador (clasificación) o un aproximador de funciones (regresión).

[2] En el entrenamiento supervisado se tienen los patrones de entrada diferenciados entre las diversas clases y las características de cada una de ellas.

Capítulo 3

La red Hopfield.

La historia de la inteligencia artificial(I. A.) es curiosa. Los primeros problemas con los que se enfrentaron los investigadores de la IA fueron problemas como el ajedrez o la demostración de teoremas, porque pensaban que en la solución de estos problemas se hallaba la esencia de la inteligencia. La visón y la comprensión del lenguaje no eran considerados problemas difíciles. Hoy en día ya se tienen programas expertos de ajedrez, así como programas expertos en realizar diagnósticos médicos, pero ningún programa puede llevar a cabo las características básicas de percepción que tiene un niño. Los investigadores de RNA reconocen que hay una falta de coincidencia evidente entre la tecnología empleada en el procesamiento de la información en una computadora estándar y la tecnología usada por el cerebro.

Además de estas tareas de percepción, la IA está comenzando a tratar ahora los principales problemas referentes a la memoria y al razonamiento de sentido común. Ya es conocida la falta de sentido común de las computadoras. Mucha gente cree que el sentido común es una consecuencia del almacenamiento masivo de conocimiento, y aún más importante, de nuestra capacidad de acceder a conocimientos relevantes rápidamente, sin esfuerzos y a su debido tiempo.

Por ejemplo, cuando se lee la descripción "gris, grande, mamífero" automáticamente se piensa en elefante y en sus características asociadas. Accedemos a la memoria mediante *contenido*. En las implementaciones tradicionales, el acceso por contenido da lugar a complicados procedimientos de búsqueda y emparejamiento. Las redes masivamente paralelas sugieren un método más eficaz.

3.1. Modelo de la red de Hopfield.

Hopfield (1982), introdujo una red neuronal que propuso como una nueva teoría de la memoria. Una red de Hopfield tiene las siguientes importantes características:

- **Representación distribuida:** Una memoria se almacena como un patrón de activación a través de un conjunto de elementos de proceso. Las memorias pueden estar superpuestas una sobre otra; las diferentes memorias se representan por diferentes patrones sobre el *mismo* conjunto de elementos de proceso.
- **Control asíncrono y distribuido:** Cada elemento de proceso toma decisiones basadas únicamente en su propia situación local. Todas estas situaciones localesse unen para alcanzar una solución global.
- **Memoria direccionable por contenido:** Se puede almacenar un determinado número de patrones en una red. Para recuperar un patrón únicamente se necesita una parte específica de él. La red encuentra automáticamente el emparejamiento más próximo.
- **Tolerancia a falla:** Aunque algunos de los elementos procesadores de la red fallasen, ésta todavía funcionará adecuadamente.

¿Cómo se alcanzan estas características? En la figura 3.1 se muestra una sencilla red de Hopfield. Los elementos de proceso, también llamados *unidades*, siempre se encuentran en uno de dos posibles estados, activos o inactivos. En la figura las unidades en negro están activas, y las unidades en blanco están inactivas. Las unidades están conectadas unas con otras por conexiones simétricas y con pesos. Una conexión con peso positivo indica que las dos unidades tieneden a activarse la una a la otra. Una conexión con peso negativo permite que una unidad activa desactive a su unidad vecina.

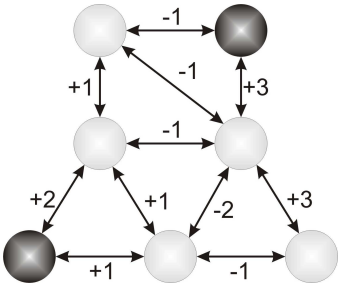


Figura 3.1: Ejemplo de una red Hopfield.

La red funciona del siguiente modo. Se elige una unidad aleatoriamente. Si alguna de sus vecinas están activa, la unidad calcula la suma de los pesos en las conexiones de estas unidades. Si la suma es positiva, la unidad se activa y si ocurre de modo contrario, se desactiva. Entonces se elige otra unidad aleatoriamente y se repite el proceso hasta que la red alcanza un estado estable, es decir, hasta que no quede ninguna unidad que pueda cambiar de estado. Este proceso se denomina *relajación paralela*. Si la red comienza a trabajar en el estado que se muestra en la figura 3.1, la unidad de la esquina inferior izquierda tenderá a activar la unidad que se encuentra por encima de ella. Esta unidad, por otro lado, tendrá a activar la unidad que se encuentra por encima de ella, pero la conexión inhibidora que produce de la unidad superior derecha aborta este intento, y así sucesivamente.

Esta red tiene únicamente cuatro estados estables distintos, que son los que se muestran en l figura 3.2. Dado un estado inicial, la red necesariamente se asentará en una de estas cuatro configuraciones[1]. La red puede verse como un "almacenador" de los patrones de la figura 3.2. La mayor contribución de las redes Hpfield es la de mostrar que dado un conjunto de pesos y un estado inicial, su algoritmo de relajación paralela en algún momento llevará a la red hacia un estado estable. No puede no existir divergencia u oscilación.

La red se puede utilizar como una memoria direccionable por contenido haciendo que las actividades de las unidades se correspondan con un patrón inicial parcial. Para recuperar un patrón, únicamente se necesita una parte de él. Entonces la red se asentará en el estado estable que mejor se empareje con el patrón parcial. También tenemos un comportamiento correcto de errores. Supóngase que se tiene la descripción "gris, grande, pez, que come plancton". Automáticamente imaginaremos una ballena, aun sabiendo que ésta es un mamífero y no un pez. Por tanto aun cuando en el estado inicial existen inconsistencias, la red de Hopfield se asentará en la solución que viole el menor número posible de restricciones que ofrecen las entradas. Los procedimientos tradicionales de emparejamiento y recuperación son menos prohibitivos.

Ahora supóngase que una unidad falla de repente haciéndose activa o inactiva cuando no debe. Esto no representaría mayor problema ya que las unidades que le rodean rápidamente volverían a ponerla en el camino correcto. Sería necesario el esfuerzo de muchas unidades erróneas para hacer que la red se saliera de un estado estable. En las redes compuestas por miles de unidades altamente interconectadas, dicha tolerancia al fallo es todavía más

[1]El estado estable en el que todas las unidades están inestables sólo puede alcanzarse si éste es el estado inicial.

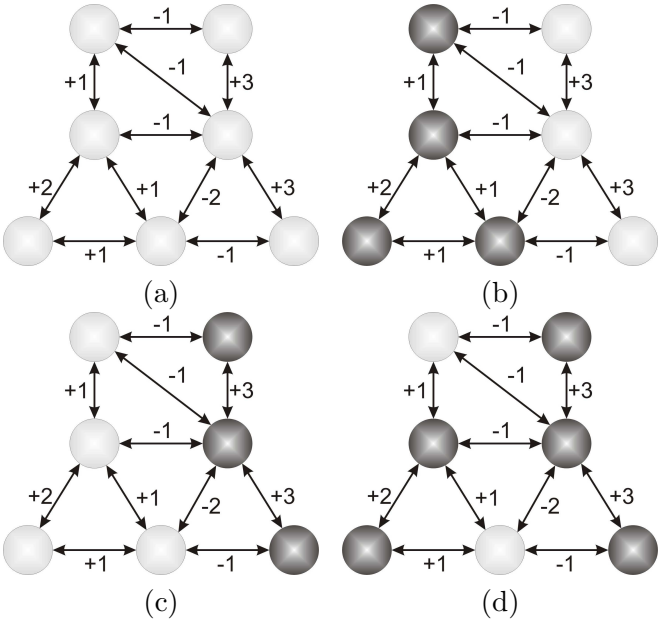


Figura 3.2: Los cuatro estados estables de un red Hopfield concreta.

patente, ya que las unidades y las conexiones pueden desaparecer completamente sin afectar de un modo adverso al comportamiento general de la red.

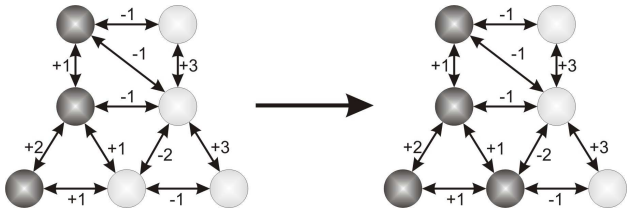


Figura 3.3: Una visión sencilla sobre lo que puede calcular una red Hopfield.

Por tanto, las redes paralelas de lementos simples pueden hacer cálculos muy interesantes. La cuestión que se plantea es: ¿Cuál es la relación entre los pesos de las conexiones de la red y los mínimos locales en los que se asista? En otras palabras, si los pesos codifican el conocimiento de una red en particular ¿cómo dicho conocimiento? Una de las características de las arquitecturas conexionistas es que su métodos de representación (denominado de pesos de

conexiones de valor real) también se presta a sí mismo para el aprendizaje automático.

Capítulo 4

El perceptron de capa simple.

La RNA más utilizada es el perceptrón de multi-capas(MLP por sus siglas en inglés), este modelo es una variante del perceptrón propuesto por Rosenblatt en 1950. El perceptrón ha generado mucho interés debido a su habilidad para aprender y reconocer patrones simples.

4.1. Modelo del perceptron.

Un perceptrón es una neurona la cual decide si una entrada pertenece a una de dos clases (denominadas A o B) como se muestra en la figura 4.1(b). El perceptrón calcula la suma ponderada de las entradas, restando el umbral, el resultado obtenido de esta operación es evaluado en la función de activación para dar origen a la salida del perceptrón. La salida de este tipo de neurona puede ser 0 ó 1. La regla de clasificación de los patrones, indica que el patrón de entrada pertenece a la clase A, cuando la salida de la neurona es 1 y a la clase B, cuando la salida es 0. Un método útil para analizar el comportamiento del perceptrón, consiste en construir una gráfica del mapa de las regiones de decisión extendidas en el espacio multidimensional creado por las variables de entrada. Estas regiones de decisión especifican cuales valores de entrada resultan en una clase A y cuales en una clase B. El perceptrón forma dos regiones de decisión separadas por un hiperplano. En la figura 4.1(b) son mostradas las regiones de decisión cuando existen dos entradas en el perceptrón y el hiperplano es representado por una línea recta en este caso. Las entradas que se encuentran por encima de la línea recta corresponden a la clase A y las entradas debajo de esta línea pertenecen a

la clase B. Como se puede observar el hiperplano de separación entre clases depende de los pesos sinápticos y del umbral.

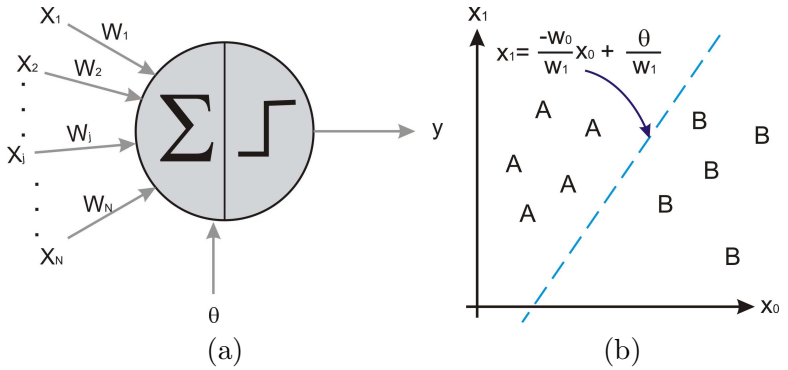


Figura 4.1: (a) Modelo de un perceptrón (b) Clasificación mediante un hiperplano

El perceptrón visto de manera general se define como, una neurona simple donde la función de red es lineal y se utiliza la función escalón como función de activación. La entrada a la neurona $x = (x_1, x_2, ..., x_N)^T$ es un vector de características en el espacio n-dimensional. La función de red $u(x)$ es la suma ponderada de las entradas definida por la ecuación 4.1 y la salida $y(x)$ se obtiene al evaluar $u(x)$ en la función de activación (ec. 4.2).

$$u(x) = \sum_{i=1}^{N} w_i x_i + \theta \tag{4.1}$$

$$y(x) = \begin{cases} 1 & u(x) \geq 0 \\ 0 & u(x) < 0 \end{cases} \tag{4.2}$$

4.2. Algoritmo de aprendizaje.

Los pesos sinápticos y el umbral en un perceptrón pueden ser fijos o adaptativos de acuerdo al uso de los diferentes algoritmos. El vector de pesos $w = (w_1, w_2, ..., w_N)$ debe ser determinado de acuerdo a los ejemplos de entrenamiento $\{(x(i), d(i)); i \in I_r\}$ y validados mediante los ejemplos de prueba $\{(x(i), d(i)); i \in I_t\}$. En los ejemplos, $d(i) (\in \{0, 1\})$ es el valor de la salida deseada de $y(x(i))$, si el vector de pesos w es ajustado de manera correcta y los índices I_r y I_t se seleccionaron adecuadamente, entonces la salida obtenida y deseada serán muy cercanas.

El algoritmo de aprendizaje (fig. 4.3) puede ser aplicado de manera iterativa, estimando el valor del vector de pesos w correcto para los ejemplos seleccionados para la fase de entrenamiento. Los pesos son estimados mediante la ecuación 4.3, donde $y(k)$ es la salida del perceptrón calculada mediante las ecuaciones 4.1 y 4.2.

$$w(k+1) = w(k) + \eta\left(d(k) - y(k)\right)x(k) \tag{4.3}$$

En la ecuación 4.3, el factor de aprendizaje $\eta\left(0 < \eta < \frac{1}{|x(k)|_{max}}\right)$ es un parámetro libre, donde $|x(k)|_{max}$ es la máxima magnitud del vector de entrada $\{x(k)\}$. El índice k indica el número de ejemplo aplicado al perceptrón de manera aleatoria. En cada iteración del algoritmo de aprendizaje, se compara la salida del perceptrón $y(k)$ con la salida deseada $d(k)$. Si ambas son iguales, entonces significa que el vector de pesos w es correcto para el ejemplo, y por lo tanto los pesos no cambiarán. Sin embargo, si $y(k) \neq d(k)$, entonces el vector de pesos w será actualizado mediante un pequeño cambio en dirección del vector de entrada $x(k)$. Si los ejemplos de entrenamiento son linealmente separables, el algoritmo de aprendizaje del perceptrón convergerá a una solución factible del vector de pesos en un número finito de iteraciones. Por otro lado, si los ejemplos de entrenamiento no son linealmente separables, el algoritmo de aprendizaje del perceptrón no convergerá a valores fijos, si el factor de aprendizaje es diferente de 0.

4.3. Aplicaciones del perceptrón.

Existen algunas dificultades al aplicar el perceptrón para resolver problemas reales de clasificación y detección de señales. Algunos de éstos problems son:

- La trasformación no lineal del vector de características, no es específica del problema.
- El algoritmo de aprendizaje del perceptrón no converge para valores fijos del factor de aprendizaje η, si los patrones de entrenamiento no son linealmente separables.
- Si el patrón de entrada es linealmente separable, entonces no se conoce el tiempo que le tomará al algoritmo de entrenamiento el converger al vector de pesos que corresponda al hiperplano necesario para separar los patrones.

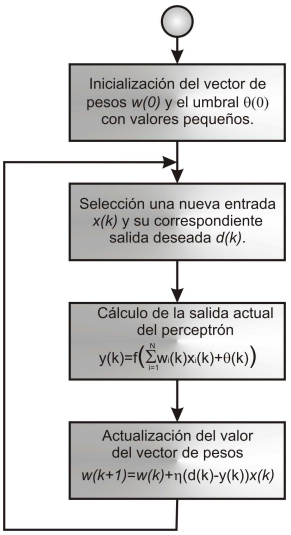


Figura 4.2: Algoritmo de aprendizaje para el perceptrón.

Capítulo 5

El perceptron de múltiples capas.

Un MLP es una modelo de RNA con propagación directa, el cual esta constituido por una o más capas entre los nodos de entrada y salida. Las capas en el MLP al menos contienen una neurona, la cual está definida por el modelo de McCulloch y Pitts. Cada neurona en el MLP tiene una función de activación no lineal y pueden ser diferentes entre cada una de las neuronas. Las funciones de activación más utilizadas en los MLP son mostradas en el apéndice A.

Un MLP supera mucha de las limitaciones del perceptrón de capa simple, pero no eran muy utilizados porque los algoritmos de entrenamiento demandaba sistemas de computo muy elevados. Esto ha cambiado recientemente con el desarrollo de nuevos algoritmos de entrenamiento y el aumento en la potencia de procesamiento mediante sistemas de computo más sofisticados. Aunque no se puede prever que los algoritmos converjan como en el perceptrón, se ha mostrado que los MLP son exitosos en muchos problemas.

5.1. Modelo de un perceptron de múltiples capas.

En la figura 5.1 se muestra la configuración típica de un MLP. Las neuronas en el MLP están organizadas en capas, nombradas como: capa oculta #1, capa oculta #2 y capa de salida. Frecuentemente al vector de entrada se les nombra capa de entrada, a pesar que esta no contengan neuronas. Las capas ocultas, son todas aquellas que se encuentran entre las entradas y la capa de salida y son numeradas de manera ascendente iniciando desde la entrada. En la figura 5.1, el índice Q indica el número de elementos en

el vector de entrada $x = (x_1, x_2, ..., x_Q)^T$, cada uno de los elementos en el vector x está relacionado con cada una de las neuronas de la capa oculta #1 por medio de un peso w_{ij}^L, donde L indica el número de la capa oculta, en este caso L es 1, i indica el número de la neurona en la capa oculta #1, donde $1 \leq i \leq M$ y j el elemento del vector de entrada. Así se pueden expresar los pesos de la capa oculta #1 mediante la matriz de pesos mostrada en la ecuación 5.1. De manera similar el umbral θ_i^1 representa el sesgo de la neurona i de la capa oculta #1 (ec. 5.2). El vector de salida de la capa oculta #1 $z^1 = \left(z_1^1, z_2^1, ..., z_i^1, ..., z_M^1\right)^T$ está definido por la ecuación 5.3, donde $f^1\left(u^1\right)$ es la función de activación de la capa.

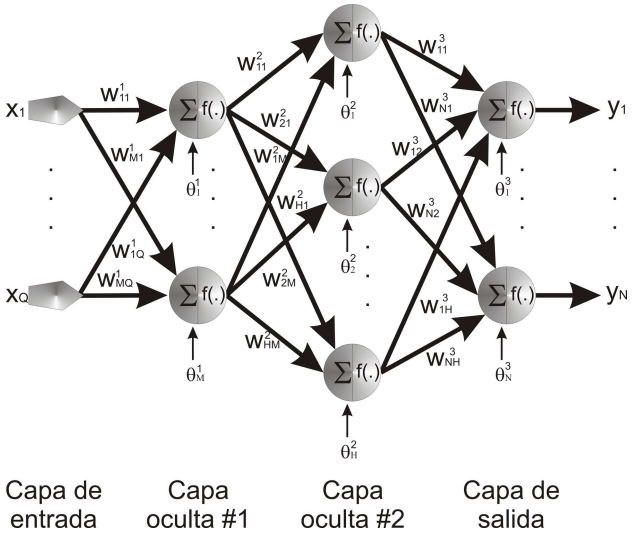


Figura 5.1: Modelo general de un perceptrón multi-capa.

$$\mathbf{W}^1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 & \dots & w_{1j}^1 & \dots & w_{1Q}^1 \\ w_{21}^1 & w_{22}^1 & \dots & w_{2j}^1 & \dots & w_{2Q}^1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{i1}^1 & w_{i2}^1 & \dots & w_{ij}^1 & \dots & w_{iQ}^1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{M1}^1 & w_{M2}^1 & \dots & w_{Mj}^1 & \dots & w_{MQ}^1 \end{pmatrix} \tag{5.1}$$

$$\theta^1 = \begin{pmatrix} \theta_1^1 \\ \theta_2^1 \\ \vdots \\ \theta_i^1 \\ \vdots \\ \theta_M^1 \end{pmatrix} \tag{5.2}$$

$$z^1 = f^1\left(\mathbf{W}^1 \times x + \theta^1\right) \tag{5.3}$$

Generalizando la ecuación 5.3 se puede calcular la salida de la capa L mediante 5.4.

$$z^L = f^L\left(\mathbf{W}^L \times z^{L-1} + \theta^L\right) \tag{5.4}$$

El vector de salida $y = (y_1, y_2, ..., y_N)^T$ del MLP mostrado en la figura 5.1 se puede obtener mediante la ecuación 5.5.

$$y = f^3\left(\mathbf{W}^3 \times \left(f^2\left(\mathbf{W}^2 \times \left(f^1\left(\mathbf{W}^1 \times x + \theta^1\right)\right) + \theta^2\right)\right) + \theta^3\right) \tag{5.5}$$

5.2. Algoritmos de aprendizaje.

En el modelo del MLP la parte más importante son los pesos, es decir, la matriz de pesos sinápticos de cada capa es la que contiene las información respectiva a la aplicación específica. El valor de cada peso es calculado mediante el algoritmo de aprendizaje de programación inversa.

El algoritmo de propagación inversa es una generalización del algoritmo de LMS, éste es una técnica de búsqueda de gradiente para minimizar la función de costo a la diferencia media cuadrática entre la salida de la red real y la deseada. La red es entrenada seleccionando inicialmente pequeños pesos aleatorios y umbrales internos y luego presentando todos los patrones de entrenamiento repetidamente. Los pesos son ajustados después de cada evento, usando información del error hasta que los pesos convergen y la función de costo es reducida a un valor aceptable. La parte esencial del algoritmo es la propagación del error de manera inversa desde la capa de salida hasta la capa de entrada.

5.2.1. Algoritmo de entrenamiento para una neurona.

Para ejemplificar el proceso de aprendizaje en un MLP, primero consideraremos el proceso en una sola neurona. La figura 5.2 muestra el modelo de una neurona, ésta está constituida por dos partes: una suma ponderada de las entradas para calcular la salida de la función de red u, y la función de activación no lineal $z = f(u)$. La salida de la red z es comparada con el valor deseado d, su diferencia, y se calcula el error $e = d - z$.

La red de la figura 5.2 tiene dos entradas $(x_1, x_2)^T$ con sus correspondientes pesos w_1 y w_2. El término θ representa el umbral interno. La función de red se calcula mediante la siguiente ecuación:

$$u = \sum_{i=1}^{2} w_i \cdot x_i + \theta = \mathbf{W} \times x + \theta \tag{5.6}$$

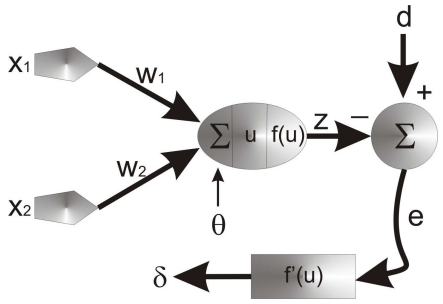


Figura 5.2: Entrenamiento de propagación inversa en una neurona.

Dado los ejemplos de entrenamiento $\{(x(k), d(k)); 1 \leq k \leq K\}$, para poder calcular el error necesario en el algoritmo de propagación inversa, primero necesitamos obtener todas las salidas $\{z(k); 1 \leq k \leq K\}$ correspondientes a los K patrones de entrada. Se utilizará una matriz de pesos $\mathbf{W}$ inicializada de manera aleatoria. La suma del error cuadrático se puede calcular mediate la ecuación 5.7.

$$E = \sum_{k=1}^{K} [e(k)]^2 = \sum_{k=1}^{K} [d(k) - z(k)]^2 = \sum_{k=1}^{K} [d(k) - f(\mathbf{W} \times x(k))]^2 \tag{5.7}$$

El objetivo del algoritmo de entrenamiento es ajustar la matriz $\mathbf{W}$ para minimizar el error E. Éste es un problema de optimización cuadrática no

lineal. Existen numerosos algoritmos de optimización no lineal que pueden resolver este problema. Fundamentalmente, éstos algoritmos utilizan una formulación iterativa:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \triangle\mathbf{W}(t) \tag{5.8}$$

En la ecuación 5.8 $\triangle\mathbf{W}(t)$ es la corrección hecha a la matriz de pesos actuales $\mathbf{W}(t)$, la cual entrega la matriz de pesos actualizados $\mathbf{W}(t+1)$.

El algoritmo más empleado en la actualización de la matriz de pesos, es el método del descendiente del gradiente, éste da la base del algoritmo de entrenamiento de propagación inversa. El método del descendiente del gradiente está basado en cuantificar la suma del error cuadrático con respecto a cada elemento de la matriz de pesos $\mathbf{W}$. Éste se calcula de la siguiente manera:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial e}\frac{\partial e(k)}{\partial z(k)}\frac{\partial z(k)}{\partial u(k)}\frac{\partial u(k)}{\partial w_i} \tag{5.9}$$

Diferenciando la ecuación 5.7 con respecto a e se obtiene:

$$\frac{\partial E}{\partial e} = \sum_{k=1}^{K}\frac{\partial\left[e(k)\right]^2}{\partial e(k)} = \sum_{k=1}^{K} 2e(k) = \sum_{k=1}^{K} 2\left[d(k) - z(k)\right] \tag{5.10}$$

De manera similar, diferenciando el error con respecto a $z(k)$ se obtiene:

$$\frac{\partial e(k)}{\partial z(k)} = \frac{\partial\left[d(k) - z(k)\right]}{\partial z(k)} = -1 \tag{5.11}$$

El siguiente paso, es encontrar la derivada de la salida con respecto a la salida de la función de red. La cual se define como:

$$\frac{\partial z(k)}{\partial u(k)} = \frac{\partial f(u(k))}{\partial u(k)} = f'(k) \tag{5.12}$$

La ecuación 5.6 se diferencia con respecto a w_i, de la cual obtenemos:

$$\frac{\partial u(k)}{\partial w_i} = \frac{\partial}{\partial w_i}\left(\sum_{j=1}^{2} w_j \cdot x_j(k) + \theta\right) = x_i \tag{5.13}$$

Sustituyendo las ecuaciones 5.10, 5.11, 5.12 y 5.13 en la ecuación 5.9 obtenemos:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}\left[d\left(k\right) - z\left(k\right)\right] f'\left(u\left(k\right)\right) x_i\left(k\right) \tag{5.14}$$

Donde $\delta\left(k\right) = \left[d\left(k\right) - z\left(k\right)\right] f'\left(u\left(k\right)\right)$, entonces la ecuación 5.14 se puede reescribir como:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}\delta\left(k\right) x_i\left(k\right) \tag{5.15}$$

$\delta\left(k\right)$ es llamado gradiente local y se define como el error $e\left(k\right) = d\left(k\right) - z\left(k\right)$ modulado por la derivada de la función de activación $f'\left(u\left(k\right)\right)$ e indica la cantidad de corrección necesaria para ser aplicada en el peso w_i en presencia de la entrada $x_i\left(k\right)$. El cambio $\triangle w_i$ es la suma de las correcciones para los K ejemplos de entrenamiento. Por consiguiente, la ecuación para la actualización de pesos es:

$$w_i\left(t+1\right) = w_i\left(t\right) + \eta\sum_{k=1}^{K}\delta\left(k\right) x_i\left(k\right) \tag{5.16}$$

En el caso en el cual la función de activación sea sigmoidal, usando la tabla 2.2, se calcula $\delta\left(k\right)$ como:

$$\delta\left(k\right) = \frac{\partial E}{\partial u} = \left[d\left(k\right) - z\left(k\right)\right] \cdot z\left(k\right) \cdot \left[1 - z\left(k\right)\right] \tag{5.17}$$

En el algoritmo de entrenamiento cada ocasión que la matriz de pesos es actualizada es llamada iteración o epoch. En esta caso, cada iteración tiene K ejemplos de entrenamiento. Se puede decir que el tamaño de una iteración es K. En la práctica, en cada iteración se tiene un tamaño desde un sólo ejemplo hasta el número total de ejemplos de entrenamiento.

5.2.2. Algoritmo de entrenamiento para múltiples capas.

En la figura 5.3, la función de red $u_j^{L-1}\left(k\right)$ y la salida $z_j^{L-1}\left(k\right)$ corresponden al $k$ésimo ejemplo de entrenamiento de la $j$ésima neurona de la $(L-1)$ésima capa. La entrada de la primer capa es $z_j^0\left(k\right) = x_j\left(k\right)$. La salida $z_j^{L-1}\left(k\right)$ es conectada a la $i$ésima neurona de la Lésima capa por medio del peso sináptico $w_{ij}^L\left(t\right)$ o para simplificar w_{ij}^L, si consideramos sólo una iteración, para realizar la deducción de la ecuación en la actualización de los pesos.

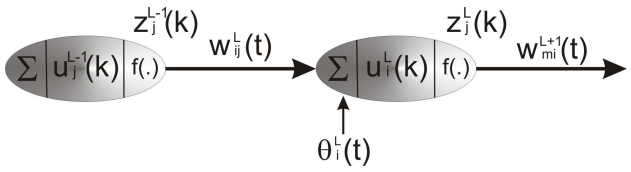


Figura 5.3: Notación utilizada para un MLP de múltiples capas.

Para derivar la ecuación de adaptación de pesos, primero es necesario generalizar la ecuación 5.15, para calcular $\partial E/\partial w_{ij}^{L}$ de la siguiente manera:

$$\frac{\partial E}{\partial w_{ij}^{L}} = -2\sum_{k=1}^{K} \delta_i^K(k) \cdot z_j^{L-1}(k) \tag{5.18}$$

En la figura 5.4, la salida $z_j^{L-1}(k)$ se puede avaluar, calculando para el $k$ésimo ejemplo de entrenamiento $x(k)$ con la matriz de pesos fija. Sin embargo, el término de error delta $\delta_i^L(k)$ se debe calcular para las capas ocultas.

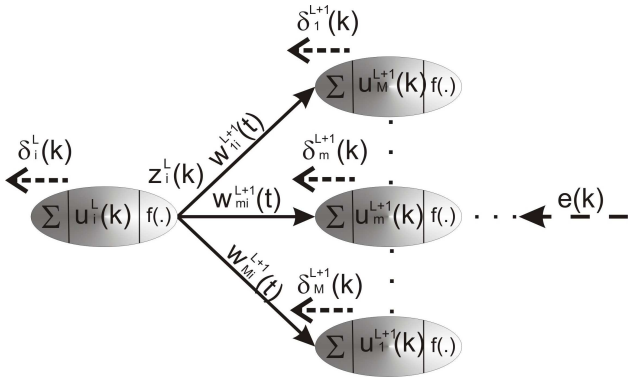


Figura 5.4: Propagación inversa del error.

El término de error delta se puede definir como $\delta_i^L(k) = \partial E/\partial u_i^L(k)$. La figura 5.4 muestra como se puede calcular $\delta_i^L(k)$ de manera iterativa mediante $\delta_m^{L+1}(k)$ y la matriz de pesos de la $(L+1)$ésima capa.

La salida $z_i^L(k)$ está conectada con las M neuronas en la $(L+1)$ésima capa. Por lo tanto, se puede decir:

$$\delta_i^L(k) = \frac{\partial E}{\partial u_i^L(k)} = \sum_{m=1}^{M} \frac{\partial E}{\partial u_m^{L+1}(k)} \cdot \frac{\partial u_m^{L+1}(k)}{\partial u_i^L(k)} \tag{5.19}$$

donde $\partial E/\partial u_m^{L+1}(k)$ se puede expresar como:

$$\frac{\delta E}{\delta u_m^{L+1}(k)} = \delta_m^{L+1}(k) \tag{5.20}$$

y $\partial u_m^{L+1}(k)/\partial u_i^L(k)$ como:

$$\frac{\partial u_m^{L+1}(k)}{\partial u_i^L(k)} = \frac{\partial}{\partial u_i^L(k)} \left[\sum_{j=1}^{N} w_{mj}^{L+1} z_j^{L+1}(k) \right] = \sum_{j=1}^{N} \frac{\partial w_{mj}^{L+1} f\left(u_j^L(k)\right)}{\partial u_i^L(k)} \tag{5.21}$$

Sustituyendo las ecuaciones 5.20 y 5.21 en la ecuación 5.19:

$$\delta_i^L(k) = f'\left(u_i^L(k)\right) \cdot \sum_{m=1}^{M} \delta_m^{L+1}(k) \cdot w_{mi}^{L+1} \tag{5.22}$$

La ecuación 5.22 es la fórmula para la propagación inversa del error, mediante esta se calcula la delta de error para la capa de salida y se propaga de capa en capa hasta la de entrada.

5.2.3. Actualización de pesos con momentum.

Mediante la delta de error, se puede modificar la ecuación 5.16 para obtener:

$$w_{ij}^L(t+1) = w_{ij}^L(t) + \eta \cdot \sum_{k=1}^{K} \delta_i^L(k)\, z_j^{L-1}(k) + \mu \left[w_{ij}^L(t) - w_{ij}^L(t-1) \right] \tag{5.23}$$

En la ecuación 5.23, el segundo término es el gradiente con respecto al peso w_{ij}^L. El tercer término es conocido como momentum. Este provee un mecanismo adaptativo para ajustar el tamaño de la actualización del peso. Cuando el vector de gradiente tiene una misma dirección en alguna iteración sucesiva, éste incrementa el ajuste del peso (ganando momentum). Cuando el vector de gradiente cambia continuamente de dirección, el momentum ayuda a minimizar el error cuadrático medio.

Existen dos parámetros libres por seleccionar en la ecuación anterior: el factor de aprendizaje η y la constante de momentum μ. Ambos parámetros

deben ser seleccionados en el intervalo $[0, 1]$. En la práctica, η toma valores pequeños, por ejemplo, $0 < \eta < 0{,}3$, y μ toma valores grandes, por ejemplo, $0{,}6 < \mu < 0{,}9$.

5.3. Implementación del modelo.

Para realizar la implementación de un MLP, se tomará como ejemplo la figura 5.5. En ésta figura se muestra un MLP de cuatro capas, con 2 elementos en la capa de entrada, 2 neuronas en la capa oculta #1, 3 neuronas en la capa oculta #2 y 1 neurona en la capa de salida.

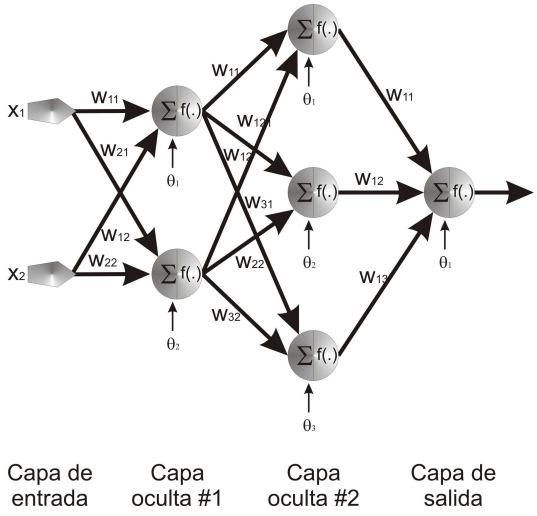


Figura 5.5: MLP de 4 capas con [2,2,3,1].

El primer paso es calcular la salida de la capa oculta #1 para el $k$ésimo ejemplo como:

$$\begin{bmatrix} z_1^1(k) \\ z_2^1(k) \end{bmatrix} = f\left(\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \times \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \right) \tag{5.24}$$

La ecuación 5.24 se puede escribir de la siguiente manera:

$$z^1(k) = f\left(\mathbf{W}^1 \times z^0(k) + \theta^1\right) \tag{5.25}$$

De manera similar se obtiene las salidas de capa oculta #2 y de la capa de salida:

$$z^2(k) = f\left(\mathbf{W}^2 \times z^1(k) + \theta^2\right) \tag{5.26}$$

$$y(k) = z^3(k) = f\left(\mathbf{W}^3 \times z^2(k) + \theta^3\right) \tag{5.27}$$

Con la salida de la red calculada, el siguiente paso es realizar la actualización de pesos, el cual se efectúa de manera secuencial mediante la ecuación 5.23. Si la salida deseada para el $k$ésimo ejemplo de entrada es $d\,(k)$, entonces $\delta^3\,(k)$ se puede obtener mediante la siguiente ecuación:

$$\delta^3\,(k) = [d\,(k) - y\,(k)] \cdot f'\left(u^3\,(k)\right) \tag{5.28}$$

Para las capas ocultas se ocupa la siguiente ecuación:

$$\delta_i^L\,(k) = f'\left(u_i^L\,(k)\right) \cdot \sum_{m=1}^{M} \delta_m^{L+1}\,(k)\, w_{mi}^{L+1} \tag{5.29}$$

La ecuación 5.29 se pude reescribir como:

$$\delta^L\,(k) = f'\left(u^L\,(k)\right) \cdot \left[\mathbf{W}^{T\ L+1} \times \delta^{L+1}\,(k)\right] \tag{5.30}$$

Mediante las ecuaciones 5.23, 5.28 y 5.30 se realiza la actualización de las matrices de pesos para cada capa.

Capítulo 6

Support Vector Machines.

6.1. Introducción.

Los Support Vector Machine(SVM) son una técnica de la teoría de aprendizaje estadístico, las cuales proveen una nueva propuesta al problema del reconocimiento de patrones. En el área de reconocimiento de patrones, los SVM han sido usados en el reconocimiento de escritura manuscrita aislada, en el reconocimiento de objetos, en la identificación de voz, en la detección de quarks, en la detección de rostros en imágenes, entre otras aplicaciones. Debido a esas aplicaciones de los SVM, éstos pueden ser comparados con propuestas para el reconocimiento de patrones, tales como las RNA. La diferencia entre éstas últimas y los SVM radica en el entrenamiento de los SVM, durante el cual siempre se encuentra un mínimo global y su interpretación geométrica simple proporciona un amplio campo para investigaciones adicionales.

En sentido estricto, una tarea de aprendizaje dada con una cantidad finita de datos de entrenamiento, alcanza su mejor desempeño cuando se realiza un balance correcto entre la exactitud lograda sobre ese conjunto particular de entrenamiento y la capacidad de la máquina, es decir, la habilidad de la máquina para aprender cualquier conjunto de entrenamiento sin error. Un claro ejemplo al respecto, lo proporciona en donde se menciona que una máquina con demasiada capacidad es como un botanista con una memoria fotográfica al cual, cuando le presentan un árbol nuevo, concluye que no es un árbol debido a que tiene un numero diferente de hojas con respecto a cualquier otro que haya visto antes; sin embargo, una máquina con muy poca capacidad es como el hermano flojo del botanista que declara que si es verde entonces es un árbol. Ninguno puede generalizar bien. La exploración

y formalización de estos conceptos ha resultado en uno de los puntos más brillantes de la teoría del aprendizaje estadístico.

6.2. Definición.

Un SVM es una maquina entrenada la cual es capaz de predecir una salida de acuerdo con una entrada dada. En un proceso de aprendizaje supervisado, donde son presentados ejemplos etiquetados, la tarea consiste en encontrar una función la cual describa la relación entre los ejemplos de entrada con los de salida. En el caso de los SVM de clase binaria, la función utilizada para predecir la salida es:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{K}(\mathbf{x_i}, \mathbf{x}) + b\right) \tag{6.1}$$

Donde $\mathbf{x_i}$, $i = 1, ..., m$ son los ejemplos de entrenamiento seleccionados llamados vectores de soporte, y $\mathbf{x}$ es el vector de entrada, la función $\mathbf{K}(\mathbf{x_i}, \mathbf{x})$, llamada kernel, es una función positiva simétrica, y_i es la etiqueta para el vector $(1, -1)$, y α_i es un peso para el vector de soporte determinado durante el proceso de entrenamiento, b es el sesgo del hiperplano. Es posible encontrar diferentes tipos de kernels:

$$\text{Lineal: } \mathbf{K}(\mathbf{x_i}, \mathbf{x}) = x_i \cdot x \tag{6.2}$$

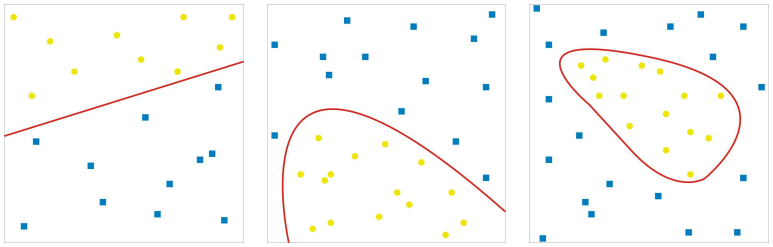
$$\text{Polinomial: } \mathbf{K}(\mathbf{x_i}, \mathbf{x}) = (x_i \cdot x + 1)^d \tag{6.3}$$

$$\text{Gaussiana: } \mathbf{K}(\mathbf{x_i}, \mathbf{x}) = e^{-\frac{||\mathbf{x_i} - \mathbf{x}||^2}{2\sigma^2}} \tag{6.4}$$

donde d es el grado del kernel polinomial y σ es la varianza de la función Gaussiana. La función polinomial y la Gaussiana son kernels no lineales, los cuales son importantes y necesarios para los casos en los que no existe una relación lineal entre los datos de entrada y las etiquetas asignadas a la salida. Así, estos kernels pueden resolver los problemas no-lineales.

6.3. Entrenamiento en los Support Vector Machine.

El proceso de entrenamiento de un clasificador SVM se deriva de la teoría de regularización:



(a) (b) (c)

Figura 6.1: Ejemplos de diferentes clasificadores (a) Lineal (b) Polinomial (c) Gaussiano

$$\underset{f \in H}{min} \frac{1}{m} \sum_{i=1}^{m} V\left(y_i, f\left(\mathbf{x}_i\right)\right) + \lambda ||f||_K^2 \tag{6.5}$$

Donde $f \in H$ es del tipo:

$$f\left(\mathbf{x}\right) = \sum_{i=1}^{m} y_i \alpha_i \mathbf{K}\left(\mathbf{x}_i, \mathbf{x}\right) + b \tag{6.6}$$

y V es la función de perdida la cual mide la exactitud de la salida predicha $f(\mathbf{x}_i)$ con respecto a la etiqueta y_i dada. Existen varios tipos diferentes de funciones de perdida, para la clasificación de los SVM. La siguiente función de perdida es usada:

$$V\left(f\left(x\right), y\right) = \left(1 - yf\left(x\right)\right)_+ \tag{6.7}$$

donde $(t)_+ = t$ si $t > 0$, y cero de otra manera.

La mayoría de la literatura acerca de SVM, estas ecuaciones son reparametrizados. En lugar del parámetro de regularización λ, la regularización es controlada via un parámetro C, definido usando la relación:

$$C = \frac{1}{2\lambda m} \tag{6.8}$$

El parámetro C da al usuario la posibilidad de escoger un costo extra para los errores. Para valores mayores de C les corresponde asignar una mayor penalidad de errores. Usando la definición, el problema de regularización se convierte en:

$$\min_{f \in H} C \sum_{i=1}^{m} V\left(y_i, f\left(\mathbf{x}_i\right)\right) + \frac{1}{2}||f||_K^2 \tag{6.9}$$

Esto nos lleva a un problema ya sea primario o dual, de los cuales ambos son programas cuadráticos convexos.

6.4. Ventajas de los clasificadores basados en SVM.

Existen varias ventajas al utilizar los SVM, la mas importante es que durante el proceso de entrenamiento, solamente pocos vectores fuera del conjunto de entrenamiento son seleccionados para convertirse en vectores de soporte. Esto reduce el costo computacional y provee una mejor generalización, lo cual proporciona otra ventaja más. Además, no existen mínimos locales en la programación cuadratica, así la solución encontrada siempre es la optima del conjunto de entrenamiento dado. Finalmente, otra de las ventajas que presentan los SVM es que la solución no depende de las condiciones iniciales como en las redes neuronales.

6.5. Los parámetros de kernel C y σ.

La elección correcta del parámetro C (y en el caso de los kernels Gaussianos además se debe elegir σ) puede convertirse en un gran reto. Geométricamente, el parámetro C controla el ancho del margen entre una clase y una no clase. Si C es muy grande el margen se convierte muy pequeño y el tiempo de entrenamiento se vuelve extremadamente largo. Por otra parte, Si C es muy pequeño, no habrá vectores de soporte sin fronteras y el termino b no podrá ser determinado. Casi todo esto es también verdadero para σ, si este es muy pequeño la generalización se vuelve muy pobre y cada vector es usado como un vector de soporte, si es muy grande el kernel además no mostrara buenos resultados. Se propone utilizar la búsqueda en una rejilla como un método directo para encontrar los valores adecuados para C y σ.

Apéndice A

Gráficas de funciones de activación.

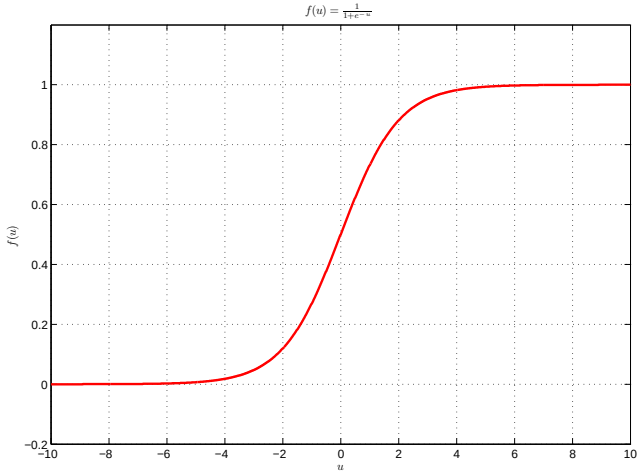


Figura A.1: Gráfica de la función sigmoidal.

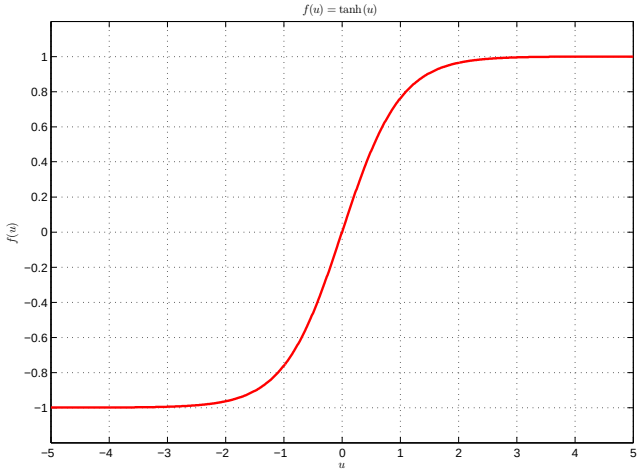


Figura A.2: Gráfica de la función tangente hiperbólica.

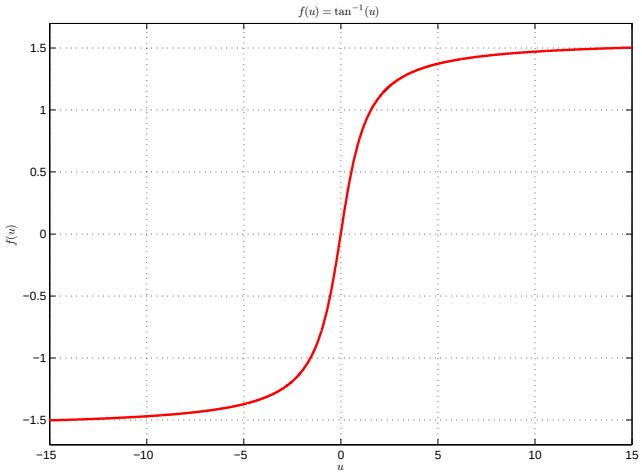


Figura A.3: Gráfica de la función tangente inversa.

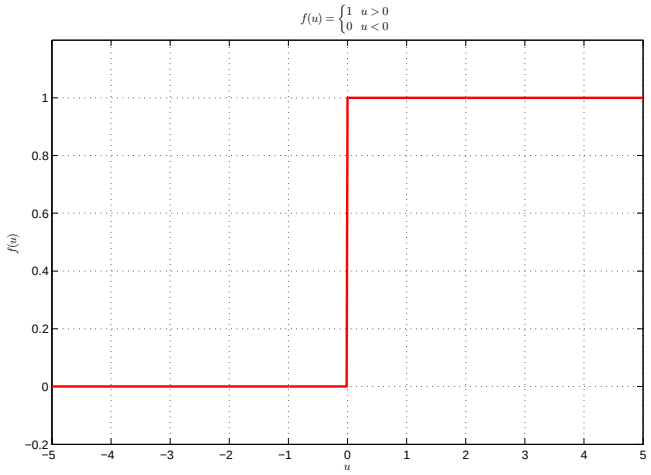


Figura A.4: Gráfica de la función escalón.

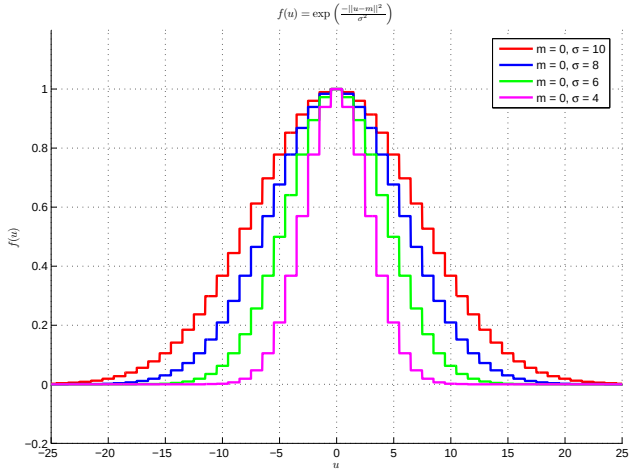


Figura A.5: Gráfica de la función gaussiana para diferentes valores de σ.

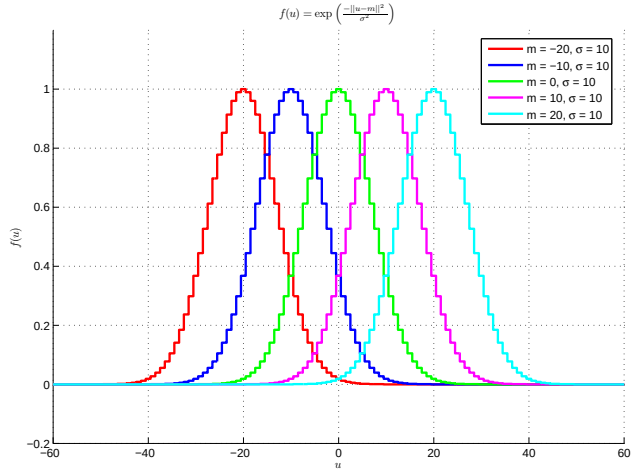


Figura A.6: Gráfica de la función gaussiana para diferentes valores de m.

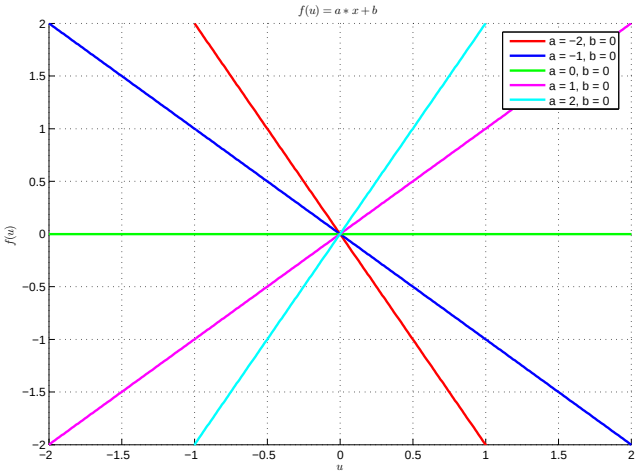


Figura A.7: Gráfica de la función lineal para diferentes coeficientes de a.

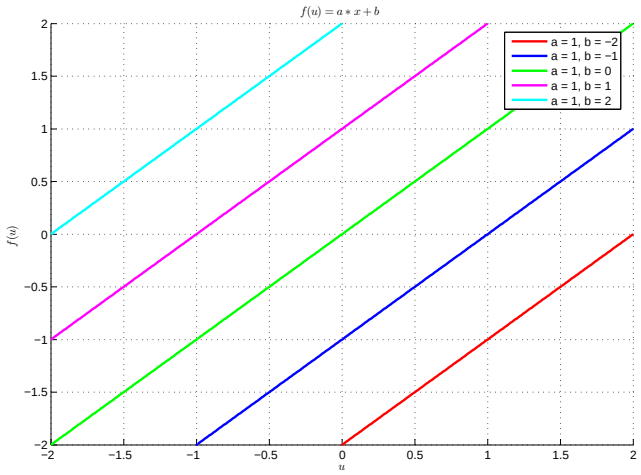


Figura A.8: Gráfica de la función lineal para diferentes coeficientes de b.